Polyphemus 1.8
User's Guide

ENPC – INRIA – EDF R&D
Meryem Ahmed de Biasi, Vivien Mallet,
Pierre Tran, Irène Korsakissok,
Damien Garaud, Édouard Debry, Lin Wu,
Marilyne Tombette, Victor Winiarek

http://cerea.enpc.fr/polyphemus/
polyphemus-help@lists.gforge.inria.fr

Contents

Chapter 1

Introduction and Installation

1.1 Polyphemus Overview

Polyphemus [Mallet et al., 2007b] is an air-quality modeling system built to manage:

- several scales: local, regional and continental scales;
- many pollutants: from non-reactive species to particulate matter;
- several chemistry-transport models;
- a bunch of advanced methods in data assimilation and ensemble forecasting;
- model integration.

Further details are available in:
Mallet, V., Quélo, D., Sportisse, B., Ahmed de Biasi, M., Debry, É., Korsakissok, I., Wu, L., Roustan, Y., Sartelet, K., Tombette, M., and Foudhil, H. (2007). Technical Note: The air quality modeling system Polyphemus. *Atmospheric Chemistry and Physics*, 7(20):5,479-5,487
This is the main reference for Polyphemus. Please cite it if you refer to Polyphemus in a publication, a talk or so.

Polyphemus is made of:

- data processing abilities (available in libraries);
- a library for physical parameterizations (library AtmoData);
- programs to compute input data to chemistry-transport models;
- chemistry-transport models (Eulerian, Gaussian and Lagrangian);
- drivers, that is, object-oriented codes responsible for driving models in order to perform, for instance, simulations and data assimilation;
- automatic generation of large ensembles, and uncertainty estimation tools;
- programs to analyze and display output concentrations (primarily based on the library AtmoPy).

Its flowchart is shown in Figure 1.1, in which three steps may be identified: (1) preprocessing (interpolations, physical parameterizations), (2) model computations (possibly with data assimilation or any other method implemented in a driver), (3) postprocessing (comparisons to measurements, statistics, visualization).

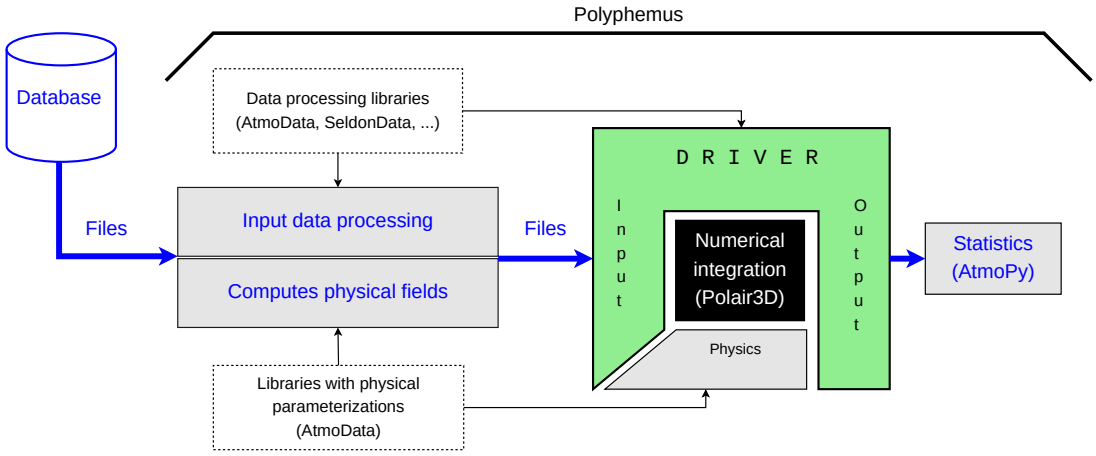


Figure 1.1: Polyphemus flowchart (preprocessing, model computations, postprocessing).

As a consequence, Polyphemus code is organized with the following directories tree:

`preprocessing`

- `bc`: boundary conditions (Mozart 2, Gocart, INCA);
- `bio`: biogenic emissions;
- `dep`: deposition velocities;
- `emissions`: pollutant emissions (EMEP);
- `ground`: ground data (land use cover, roughness);
- `ic`: initial conditions (Mozart 2);
- `meteo`: meteorological data (ECMWF, WRF and MM5, including cloud attenuation and vertical diffusion);

`processing`: subdirectories where to find programs for simulations and data assimilation with related examples of configuration and data files;

`postprocessing`: programs for comparisons to measurements;

- `water_plume`: liquid water diagnosis in a plume;
- `optics`: computation of aerosol optical parameters;
- `ensemble`: ensemble forecasting;

`include`

- `Talos`: C++ library to manage configuration files (used everywhere in Polyphemus), dates and string processing;
- `SeldonData`: C++ library to perform data processing (interpolations, input/output operations);

`AtmoData`: C++ and Fortran library for physical parameterizations and atmospheric data processing;

`atmopy`: AtmoPy is a Python library for statistical analysis and visualization;

`common`: mostly, functions used to parse and manage the arguments of preprocessing programs;

`models`: chemistry-transport, Gaussian and Lagrangian models to be used by the drivers;

`modules`

- `common`: base modules from which transport, chemistry and aerosol modules derive;
- `transport`: numerical schemes for advection and diffusion;
- `chemistry`: chemical mechanisms;
- `aerosol`: chemical mechanisms for aerosol species;

`driver`

- `assimilation`: drivers for data assimilation;
- `common`: a base driver from which all drivers are derived;
  - `observation`: observation managers for data assimilation (ground observations and simulated observations);
  - `optimization`: optimization algorithms;
  - `output_saver`: modules to save the results of a simulation;
  - `perturbation`: management of model perturbations (Monte Carlo);
- `local`: drivers for local scale applications;
- `uncertainty`: drivers that generate perturbed input data;

`utils`: useful tools, mostly to get information on binary files;

`ensemble_generation`: tools useful to ensemble generation.

Polyphemus is an open source software distributed under the GNU General Public License. It is available at http://cerea.enpc.fr/polyphemus/ or at http://gforge.inria.fr/projects/polyphemus/. Polyphemus development and support team can be contacted at polyphemus-help@lists.gforge.inria.fr.

1.2 Requirements

1.2.1 Operating Systems and Compilers

Polyphemus is designed to run under Unix or Linux-based systems. It should be able to run under Windows. AtmoPy has been tested under Windows and the Eulerian model Polair3D has been compiled with Microsoft Visual Studio.NET 2003. There is no obvious reason why other parts of Polyphemus should not work under Windows.

Polyphemus is based on three computer languages: C++, Fortran 77 and Python. There are also a very few lines of C.

Supported C++ compilers are GNU GCC (G++) 3.2, 3.3, 3.4, 4.1, 4.2, 4.3 and 4.4. GNU GCC 2.x series is too old to compile Polyphemus. Intel C++ compiler (ICC, versions 7.1, 8.0 and 9.1) should work. Any other decent C++ compiler (compliant with the standard) should compile Polyphemus. If not, please report to polyphemus-help@lists.gforge.inria.fr.

Corresponding Fortran compilers are acceptable: GNU G77 3.2, 3.3 and 3.4 and GNU GFortran 4.0, 4.1, 4.2, 4.3 and 4.4 (take care: GFortran 4.0 and 4.1 are rather slow according to our tests, you had better to install more recent versions), and Intel Fortran compilers IFC 7.1, IFORT 8.0 and IFORT 9.1.

Python versions 2.3 to 2.5 are supported.

1.2.2 External Libraries and Python Modules

With regard to software requirements, below is a list of possible requirements (depending on the programs to be run):

- the C++ library Blitz++ (http://www.oonumerics.org/blitz/): versions 0.6, 0.7, 0.8 and 0.9 are supported. Note that your compiler may exclude a few versions.
- Blas/Lapack (compiled libraries): any recent version.
- NewRan: C++ library for generation of random numbers, from version 2.0. For installation of NewRan, see Section 1.3.3.
- NetCDF (compiled libraries and headers): C++ library, any version from series 3.x should work.
- NumPy: any recent version. Make sure that your versions of NumPy and Matplotlib (see below) are compatible.
- Matplotlib: any recent version and corresponding pylab version (usually, pylab is included in Matplotlib package). It is recommended to install the corresponding version of Basemap in order to benefit from AtmoPy map-visualizations. Basemap is a toolkit available on Matplotlib website (http://matplotlib.sourceforge.net/), but usually not included in Matplotlib package.
- SciPy: any recent version.
- WGRIB: see Section 1.3.4.

All of them are open source software. Requirements are shown in Table 1.1.

NewRan is not included in Table 1.1 because it is only needed if one performs data assimilation or stochastic Lagrangian simulations. Similarly, WGRIB is only needed for preprocessing programs `preprocessing/meteo/meteo` and `preprocessing/meteo/attenuation` to work and has not been included in Table 1.1.

1.2.3 Parallel Computing

The main Polyphemus modules have been parallelized and support the following parallel computer memory architectures:

- shared memory (with the specification OpenMP 2.5);
- distributed memory (with the standard MPI-1.1);
- hybrid distributed-shared memory (with both openMP and MPI).

They correspond to different use cases:

Table 1.1: Polyphemus requirements.

	Blitz++	Blas/Lapack	NetCDF	NumPy	Matplotlib	SciPy
preprocessing						
/bc	X		X			
/bio	X					
/dep	X					
/emissions	X					
/ground	X					
/ic	X		X			
/meteo	X					
processing	X	X				
postprocessing				X	X	X
/water_plume	X					
/optics	X					
include						
/atmopy				X	X	X

- parallelized modules of advection (with a splitting method), diffusion, chemistry (RACM) and aerosol can run on a cluster of multi-core nodes where an MPI library is installed (for instance, LAM/MPI or Open MPI);
- the same modules can be parallelized with OpenMP if your compiler supports it (for instance, GNU GCC posterior to 4.2 and Intel compiler 9.1);
- if you can not install, on your system, any MPI library or any compiler suite supporting openMP, you might still exploit some parallelism within the aerosol modules. Indeed, they supports kind of multi-threading features taking advantage of multi-core POSIX platforms (see Section 6.3.2 for details).

Which parallelization do fit to your needs?

We decided to give you the choice between several parallel computing alternatives so that there are more chances one will fit your specific needs:

- MPI is a must have if you want to perform your parallel job on a cluster of processors that don't share their memory. But, it can do more. Indeed, it can also take advantage of multi-core processors, that is processors made of multiple cores that do share their memory. It is therefore our most versatile alternative for parallel computing. Its main drawbacks when comparing with the OpenMP alternative are its increased need for memory (each core might duplicate the whole memory of the job) and its application that is a little less direct and simple.
- If your compiler supports OpenMP and your computing platform is limited to a unique multi-core machine, then OpenMP could be your best pick. It is indeed as simple to compile and run an OpenMP-parallelized program than its serial counterpart. Nevertheless, it is limited to shared memory architectures: for instance, it could not run on more than one node of a standard cluster.
- The so-called hybrid OpenMP/MPI alternative looks advantageous as it combines the strengths of shared and distributed parallel models: it is not limited to a unique multi-core processor as OpenMP is and it does not need as much as memory as MPI do. You

can then give it a try if you are lucky enough to get both installed in your environment and you feel too limited with the memory available in your hardware.

Performance gains

On our computing platform, both three alternatives gave performances quite equivalent. At least, performance depends on your compilers and your hardware but it depends also on the job you are submitting.

You have then to be aware of two things in Polyphemus to better exploit parallelism:

- parallelism operates below the timestep level. The shorter the computation of a timestep is, the more the overhead induced by parallelism will weight, then the least parallelism will offer speed-up;
- the computation domain is primarily partitionned along the X-axis, sometimes along the Y-axis. Therefore, the number of core processing unit you use, should not be larger than the number of cells along the X-axis.

1.3 Installation

1.3.1 Main instructions

As soon as libraries and compilers are available, Polyphemus is almost installed. First, extract Polyphemus sources to a given directory. Polyphemus is usually distributed in a `.tar`, `.tgz`, `.tar.gz` or `.tar.bz2` file. These files are extracted with one of these commands:

```
tar xvf Polyphemus.tar
tar zxvf Polyphemus.tgz
tar zxvf Polyphemus.tar.gz
tar jxvf Polyphemus.tar.bz2
```

Polyphemus programs must be compiled by the user when needed. `SConstruct` files as well as make files (`makefile`) are provided so that program compilation should be easy.

SCons is a seductive and powerful alternative to `make` based on the Python language (http://www.scons.org/). You may not install it on your system as it is locally installed within the Polyphemus package you downloaded.

Depending on the paths to the libraries and maybe your compilers, you might need to slightly modify the `SConstruct` files or make files. Note that SCons should detect by itself the suitable compilers available on your platform.

Then, one may compile the program `meteo.cpp` in this way:

```
cd Polyphemus/preprocessing/meteo
../../utils/scons.py meteo
```

Indeed, SCons is installed in `Polyphemus/utils/`. It could be interesting to install SCons on your system or locally, so that you would just launch:

```
scons meteo
```

A simple way to achieve such a local installation if you work on a GNU/Linux platform is to modify the configuration file of your shell program. For example, with `bash`, you could edit your `.bashrc` to define `scons` as an alias for `scons.py` and add the path to `Polyphemus/utils/` in the `PATH` environment variable:

```
alias scons='scons.py'
PATH=.:~/Polyphemus/utils:$PATH
```

If you still prefer to work with `make` (although SCons is recommended instead) and the GNU compiler suite is available on your computer:

```
make -f makefile.gcc meteo
```

Then the program `meteo` is compiled and can be run. Launch `scons.py` or `make` in order to compile all programs in a given directory.

Further explanations about the compilation are shown in Section 2.3. **It is highly recommended to read them, at least if you experience any problem.**

1.3.2 AtmoPy

A special step is required with the Python library AtmoPy. This library makes calls to a C++ program in order to parse configuration files. Follow the steps below to have AtmoPy fully installed:

```
cd Polyphemus/include/atmopy/talos
../../../utils/scons.py
```

Instead of SCons, you can also perform the compiling job manually:

```
g++ -I../../Talos -o extract_configuration extract_configuration.cpp
```

You may replace g++ with any supported compiler (see Section 1.2).

1.3.3 NewRan

The library NewRan is required for Kalman algorithms (RRSQRT and ensemble), for Monte Carlo simulations, for adjoint-model validation to generate random numbers and for the Lagrangian stochastic model. It should be installed in `include/newran/`. But if you decided to use SCons, you just can skip this section and install it by typing from `include/newran`:

```
../../utils/scons.py -f SConstruct_newran
```

If you are still loyal to `make`, please go on.

Download NewRan from http://www.robertnz.net/download.html (or type "NewRan" in search engine). At the time these lines are written, NewRan 3.0 beta is available at http://www.robertnz.net/ftp/newran03.tar.gz. The following commands work with NewRan 3.0 beta (released 22 April 2006); there may be slight changes with other versions.

Go to directory `include/newran/`, expand NewRan in it:

```
cd Polyphemus/include/newran
wget http://www.robertnz.net/ftp/newran03.tar.gz
tar zxvf newran03.tar.gz
```

If everything is fine, you should have a file called `include/newran/newran.h`. Next, you have to edit `include/newran/include.h` and uncomment the line:

```
//#define use_namespace                    // define name spaces
```

That is, remove the first two slashes:

```
#define use_namespace                    // define name spaces
```

Compile the library (here, with GNU C++ compiler):

```
make -f nr_gnu.mak libnewran.a
```

This should create `include/newran/libnewran.a`. To complete the installation, you have to create a directory where the seed values are stored, for instance:

```
mkdir ~/.newran
cp fm.txt lgm.txt lgm_mix.txt mother.txt mt19937.txt multwc.txt wh.txt ~/.newran/
```

Recall the path to your seed directory since this is an entry of a configuration file (`processing/assimilation/perturbation.cfg`).

1.3.4 WGRIB

WGRIB is a library used to decode GRIB first edition files. It is only necessary if you use ECMWF meteorological fields, so for programs `meteo.cpp` and `attenuation.cpp`. If you use meteorological data from MM5 or WRF, you do not need this library.

WGRIB homepage can be found at:
http://www.cpc.ncep.noaa.gov/products/wesley/wgrib.html.

As of version 1.7 of Polyphemus, the source code of WGRIB is directly included in the distribution package so that you shouldn't need to download and install it anymore.

Please notice that we just included the version v1.8.0.12o. You might find more recent versions of WGRIB at

`ftp://ftp.cpc.ncep.noaa.gov/wd51we/wgrib/wgrib.tar`

but they might not be compatible with Polyphemus.

1.3.5 ISORROPIA

`ISORROPIA` (Nenes et al. [1998]) is an aerosol thermodynamics module which is necessary for aerosol module `Aerosol_SIREAM_SORGAM` to work. It is only needed if there are aerosol species involved.

In that case, you will need *sources* for `ISORROPIA`. You can obtain them from its home page http://nenes.eas.gatech.edu/ISORROPIA/. The version of `ISORROPIA` that is supported in Polyphemus is 1.7 (released on 2009-05-27).

After you obtained and extracted the files, you have to put the files in directory `include/isorropia` and rename them as follows:

- `ISOCOM.FOR` as `isocom.f`,
- `ISOFWD.FOR` as `isofwd.f`,
- `ISOREV.FOR` as `isorev.f`
- `ISRPIA.INC` as `isrpia.inc`

If you want to use the module `Aerosol_SIREAM_SORGAM` parallelized with openMP, you should also apply the related patch:

```
cd include/isorropia
patch -p1 < ../modules/aerosol/isorropia.patch_v1.7_2009-05-27
```

1.3.6 ISORROPIA_AEC

Aerosol module `Aerosol_SIREAM_AEC` requires a modified version of `ISORROPIA`. You are advised, if you need it, to copy `include/isorropia` (created and patched as explained in Section 1.3.5) as `include/isorropia_aec` and apply to it the patch `include/modules/aerosol/isorropia_aec.patch_vXXX` where `XXX` is the supported version of `ISORROPIA`:

```
cd include
cp -r isorropia/ isorropia_aec
cd isorropia_aec/
patch -p1 < ../modules/aerosol/isorropia_aec.patch_v1.7_2009-05-27
```

If the patch does not work, you might have a different release of `ISORROPIA 1.7`. You can then try with `include/modules/aerosol/isorropia_aec.patch_v1.7` that was related to a previous release of `ISORROPIA 1.7`. It is also possible that the encoding of some of the files in `include/isorropia_aec` is not adapted to your filesystem. Check it and modify it if necessary.

1.3.7 Fortran Subroutines

Before reading further, please note that SCons should handle automatically the problem described below. If you plan to use make instead of SCons, you are advised to read what follows carefully.

Linking a Fortran code with C++ may raise a problem when the name of a Fortran subroutine contains an underscore. In this case, the subroutine identifier (in the compiled object) may be named with two underscores at the end, instead of one (as for other Fortran subroutines). This is a strange convention that appears in G77 (but no more in GFortran which replaces G77).

In Polyphemus there are tests to deal with this. Most of the time the tests will succeed. But if you use untested compilers or if you mix compilers from different packages, you may have undefined symbols at link stage, that is, something like:

```
polair3d.o(.gnu.linkonce.t.[...]
In function ‘Polyphemus::DiffusionROS2<double>[...]
: undefined reference to ‘diff_x_’
```

In that case, Polyphemus tests have failed. But there is an easy way to fix that, be it with SCons or `make`: you need to slightly change the compilation options.

With SCons, add the option `_=1` on the command line that calls SCons. If it does not work, try the other option `_=2`.

```
../../utils/scons.py _=2 polair3d
```

In the makefile, add to the compiler flag (`CCFLAGS`) the option: `-DPOLYPHEMUS_SINGLE_UNDERSCORE`. If this does not work, then add *instead* `-DPOLYPHEMUS_DOUBLE_UNDERSCORE`. If you change compilation options, you should use `make cleanall` before compiling so that each part of the code is compiled with the same options.

If this still does not work, it is likely that your problem is elsewhere. Contact Polyphemus development team (polyphemus-help@lists.gforge.inria.fr) if you need help.

Chapter 2

Using Polyphemus

2.1 Remark

In configurations files, in output logs, and so on, indices start at 0 (as in C++ and Python, not at 1 as in Fortran).

2.2 Guide Overview

Now that you managed to complete the intallation of Polyphemus, you should be eager to know what you finally got and how you could use it. Here are some guidelines that might help you finding your way in this document and save your time. You are also warmly advised to take advantage of the test cases or training sessions we keep up-to-date on the Polyphemus website. Nothing will replace practical training.

Where are the programs?

First of all, you might want to select the processing program that is of interest for your own work. We hope you will find it among off-the-shelf programs we provided in one of the subdirectory of `processing`. Program source files can be recognized thanks to their file extension: `*.cpp`.

What are they doing?

You can get an idea of what a processing program does by looking at its content so that you can identify the model with its related driver and modules.

For instance, edit `processing/photochemistry/polair3d.cpp`:

```
int main(int argc, char** argv)
{
...
  typedef Polair3DChemistry<real, SplitAdvectionDST3<real>,
    DiffusionROS2<real>,
    Photochemistry<real> > ClassModel; (1)

  BaseDriver<real, ClassModel, BaseOutputSaver<real, ClassModel> >
    Driver(argv[1]); (2)

  Driver.Run();
```

```
...
}
```

This gives you precious informations about the objects used in the program:

(1) tells you the model used is `Polair3DChemistry`. It includes the following terms (or modules), `SplitAdvectionDST3`, `DiffusionROS2` and `Photochemistry`.

(2) indicates that the driver of the model is `BaseDriver` and that the output will be managed according to `BaseOutputSaver`.

In order to know how all these objects work, read their description in the related sections of the guide. For the example given above:

- section 5.6 Polair3DChemistry of chapter 5 Models;
- sections 6.1.2 SplitAdvectionDST3, 6.1.4 DiffusionROS2, 6.2.1 Photochemistry of chapter 6 Modules;
- section 4.1 BaseDriver of chapter 4 Drivers;
- subsection 4.9.1 BaseOutputSaver of section 4.9 Output Savers.

You now have all the elements you need to choose the processing program that fits your needs.

How to set up a simulation?

Section 2.6.4 on mandatory data for models along with the sections devoted to the related preprocessing programs of chapter 3 should help you figure out where to find the raw data you need and how to preprocess them.

It will then be time to compile the programs you have to use (see section 2.3). For instance, in `processing/photochemistry` to compile `polair3d.cpp`, type:

```
../../utils/scons.py polair3d
```

Afterwards, you will have to edit their related configuration files. In the directory of the program, you should find an example of configuration files from where to start:

- pre or postprocessing program have usually two configuration files. For instance, program `preprocessing/meteo` uses `general.cfg` and `meteo.cfg`.
- Processing programs have three of them. For instance, `processing/photochemistry/polair3d` uses `racm.cfg`, `racm-data.cfg` and `racm-saver.cfg`.

Section 2.4 might be very helpful along with appendix D for the lexical references. As for the description of the fields that control configuration, look at:

- the section documenting the program if it is a pre or postprocessing program;
- the sections documenting the objects of the programs, be it a driver, an ouptut saver, a model or a module as previously described. Indeed, all the fields that are specific to control the behaviour of a given object should be provided in the section it is related to and possibly in sections of objects from which it is derived.

At this point, the parts of section 2.6 you have not read yet become a must read.

How to run a program?

For preprocessing or postprocessing programs, examples of command line are usually given in their related sections. For preprocessing, sections 2.5 and 3.2 should provide you with additional details. As for processing programs, you will understand it quickly through an example.

Let's suppose you adopted the directory tree of 2.6.1 and you want to launch a simulation with `processing/photochemistry/polair3d`. From the directory `MyStudy`, if you did not change the name of the main configuration file `racm.cfg`, just type:

```
../Polyphemus-{version}/processing/photochemistry/polair3d config/racm.cfg
```

And then?

Once done, section 2.7 will suggest some ways to check your results and chapter 7 will guide you through the postprocessing tasks.

2.3 Compiling the Programs

Polyphemus programs are supposed to be compiled with SCons or, second choice, with make. SCons is the recommended way to compile the programs: it is more portable and autonomous, it has much more options and, if you intend to write new programs, it should ease your experience. Take notice that, as of version 1.6 of Polyphemus, makefiles are no longer supported. They are provided as is and you should check them and might have to correct them before using them.

2.3.1 Compiling with SCons

Where is SCons?

Polyphemus includes a version of SCons in `utils/` – the command is `utils/scons.py`. So you can use this version of SCons, or you can install SCons on your system (which is easy and which may be more convenient in the long term). Version 0.98.5 or greater is required. In the sequel, it is assumed that SCons is launched with the command `scons`, which you may replace with `Polyphemus/utils/scons.py` if you rely on the version distributed with Polyphemus.

Compiling

In most Polyphemus directories, you should be able to compile simply by calling SCons:

```
scons
```

or, to build a given program (say, `meteo`):

```
scons meteo
```

This is relevant for any directory where there is a C++ program to compile: `processing/decay/`, `processing/siream-sorgam/`, `preprocessing/meteo/`, `postprocessing/optics/`, ...

SCons is supposed to find the compilers and to properly determine all dependencies, on any platform. If you experience anyway problems, the course of action is:

1. try to help SCons with command line options, or by filling the `SConstruct` file; see below;
2. contact the Polyphemus mailing list polyphemus-help@lists.gforge.inria.fr.

Options

In Polyphemus, SCons compilation comes with many enjoyable options, which may increase your productivity (especially if you are developing) and which can help you solve some problems.

There are two ways to activate the options: through the command line or through some variables set in the local `SConstruct` file.

Command Line Options A command line option, say `cpp`, is introduced this way:

```
scons cpp=my_compiler meteo
```

which tells SCons to compile `meteo` with the C++ compiler `my_compiler` (instead of the default C++ compiler which SCons finds automatically on our system). Of course, if you want to compile all your programs with `my_compiler`, you launch:

```
scons cpp=my_compiler
```

Below is a list of all supported command line options (in the column of all possible values, the first value is the default value):

Option	Possible values	Explanations

`debug`	`0`, `-1`, `1`, `2`	Debug level:
		`-1` for no option related to the debug level
		`0` (default) for optimization with `-O2`
		`1` for debugging mode with `-g`
		`2` for optimized debugging mode with `-g -O2`

`debug_cpp`	`0`, `-1`, `1`, `2`	Same as `debug`, but only for C++. It overwrites `debug`.
`debug_fortran`	`0`, `-1`, `1`, `2`	Same as `debug`, but only for Fortran. It overwrites `debug`.
`line`	`no`, `yes`	Should the compilation lines be shown?
`c`		The C compiler.
`cpp`		The C++ compiler.
`fortran`		The Fortran compiler.
`link`		The linker.
`_`	`1`, `0`, `2`	Number of underscores at the end of the symbols of compiled Fortran routines whose names contain at least one underscore. If set to `0`, Polyphemus tries to guess from the C++ compiler version.
`mode_cpp`	`strict`, `permissive`	`strict` compiles C++ with options `-Wall -ansi -pedantic`, only if G++ is used. `permissive` compiles C++ without restrictive options.
`mode_fortran`	`permissive`, `strict`	`strict` compiles Fortran with options `-Wall -pedantic`, only if G77 or GFortran is used. `permissive` compiles Fortran without restrictive options.
`flag_cpp`		Additional C++ compilation flags.
`flag_fortran`		Additional Fortran compilation flags.
`flag_link`		Additional link flags.
`openmp`	`no`, `yes`	Should the parallelizing library openMP be used? (for models including modules supporting the openMP parallelization).
`flag_openmp`		Compilation flag for openMP.
`mpi`	`no`, `yes`	Should the parallelizing library MPI be used? (for models including modules supporting the MPI parallelization).

`chemistry`	`racm`, `racm2`, `cb05`	Chemistry mecanism used in the aerosol modules.
`nacl`	`no`, `yes`	Should the thermodynamics of the aerosol modules include NaCl? (for models including an aerosol module).

Variables in `SConstruct` More variables may be changed in the `SConstruct` file itself. Note that `SConstruct` files are Python scripts, sensitive to case. Have a look at `processing/photochemistry/SConstruct`. In this `SConstruct` file, several variables are included. They can be defined as strings, e.g.,

```
cpp_compiler = "my_compiler"
```

or as list of strings, e.g.,

```
include_path = ["include/Talos", "include/SeldonData"]
```

or a list of strings embedded in a string, e.g.,

```
include_path = "include/Talos include/SeldonData"
```

or, equivalently,

```
include_path = """include/Talos
                  include/SeldonData
                  """
```

If you do not want to define a variable, remove the line where it is defined or write:

```
linker = None
```

You must at least have the variables `polyphemus_path` and `include_path` set. `polyphemus_path` is the relative or absolute path to Polyphemus (e.g., `/home/user/src/Polyphemus/`, do not forget the last slash).

`include_path` is a list of paths where the dependencies (that are not installed in the system directories, not in the environment variables `CPATH` and `CPLUS_INCLUDE_PATH`) lie. So, if you install a library yourself, you may need to put the path to the library in this variable. For instance, if you install Blitz++ from the sources and not to the system directories (because you have not the root privileges), you have two options:

(recommended) 1. put the path to Blitz++ directory in the environment variable `CPATH` (or `CPLUS_INCLUDE_PATH`) and put the path to the compiled Blitz++ library in the variable `LD_LIBRARY_PATH` (or `LIBRARY_PATH`).

(alternatively) 2. put the path to Blitz++ directory in `include_path` and put the path to the compiled Blitz++ library in the variable `library_path`.

See below the list of supported variables. All variables but `polyphemus_path` and `include_path` are optional, which means that these variables may be omitted from the file or may be set to `None`. The last column ("Command line") shows the name of the corresponding command-line option, if any. In case both a command line option and a variable are set, the command line option overwrites the variable in the `SConstruct` file.

Variable	Content	Command line

polyphemus_path	Path to Polyphemus, i.e., the directory where one finds preprocessing, processing, utils, CREDITS, ...	
include_path	Path(s) to all dependencies not available in the system directories, in CPATH or in CPLUS_INCLUDE_PATH	
c_compiler	Name of the C compiler	c
cpp_compiler	Name of the C++ compiler	cpp
fortran_compiler	Name of the Fortran compiler	fortran
linker	Name of the linker	link
library_path	Path(s) to compiled libraries not available in the system directories, in LIBRARY_PATH or in LD_LIBRARY_PATH	
exclude_target	List of targets to ignore	
exclude_dependency	Files to ignore in the directories of include_path, described by regular expressions (you need to know what you are doing)	
flag_cpp	Additional C++ compilation flags.	flag_cpp
flag_fortran	Additional Fortran compilation flags.	flag_fortran
flag_link	Additional link flags.	flag_link

Note About the Libraries Search

SCons does not search all the time for the libraries. It caches the results. The first time, you may read:

```
Checking for C library blas... yes
Checking for C library gslcblas... no
```

And the second time:

```
Checking for C library blas... (cached) yes
Checking for C library gslcblas... (cached) no
```

If you want to force SCons to search again for the compiled libraries (because you have just installed one of them, or because you logged in another computer with a different installation), launch SCons with option `--config=force`.

2.3.2 Compiling with make

As of `make`, you might have to update the related `makefile`s by yourself because we don't maintain them anymore! Edit the makefiles to change the compiler. The main variables are the C++ compiler `CC`, the Fortran compiler `F77`, the linker `LINK`, and maybe the libraries `LIBS` and the include paths `INCPATH`.

2.3.3 Compiling for Parallel Computing

The following modules can support an openMP and/or an MPI-parallelization:

- the chemistry modules `Photochemistry` and `ChemistryRADM`,
- the advection module `SplitAdvectionDST3`,

- the diffusion module `DiffusionROS2`,

- the aerosol modules `Aerosol_SIREAM_SORGAM` and `Aerosol_SIREAM_AEC`.

Among the drivers, apart from `BaseDriver` who supports both the openMP and the MPI-parallelization, support for MPI is now available with `MonteCarloDriver` and `OptimalInterpolationDriver`.

This list will grow in the versions to come, so you had better to stay tuned if you are interested. Whatever, numerous programs given as examples in `processing/` can already take advantage of this feature if they make use of at least one of these modules (see Section 2.5.3).

Still, depending on your own context (see Section 1.2.3), you might want to use openMP alone, MPI alone or both openMP and MPI simultaneously. Here are the recipes that will get your programs built the way you want.

OpenMP alone

At first, let us consider the case where you just want an OpenMP parallelization for `polair3d` in `processing/photochemistry`. Using SCons, from `processing/photochemistry`, just type:

```
../../utils/scons.py openmp=yes
```

If you are lucky enough, you might be done!

If not, it means perhaps that you are using a compiler whose OpenMP flag is unknown to our SCons script. You might then take a look at your compiler's documentation in order to identify the related compilation flag. For instance, with PGI as the default compiler suite recognized by SCons, you should type from `processing/photochemistry`:

```
../../utils/scons.py openmp=yes flag_openmp=mp
```

Note 1: As adapting the `makefile` would be trickier, it could be therefore the compelling argument to switch to SCons...

Note 2: Be warned that the aerosol modules programming style do not fit with the Intel implementation of OpenMP at least in the version 9.1. Even with one core and no parallelizing directives, results differ! You are then advised not to use the Intel compiler suite with OpenMP if you need the aerosol modules.

MPI alone

We tried to make the building with MPI as simple as the one with OpenMP. From `processing/photochemistry`, type:

```
../../utils/scons.py mpi=yes
```

How could it be easier?

No way. But you could nevertheless encounter some little difficulties. For example, if the C++ compiler associated to mpiCC is g++-4.xx or posterior (you might get to know it with the command `mpiCC -showme`) whereas the Fortran compiler associated to mpif77 is g77-3.xx, you will have to deal with the GCC "underscore" problem (for further details, see 1.3.7). The command line shall be modified to:

```
../../utils/scons.py mpi=yes _=2
```

If you use `make`, you will have to edit the related `makefile.gcc-4` and change the variables `CC, CFLAGS, F77, FFLAGS` and `LINK`. Depending on your version of mpiCC, it is possible that you might need to add `-fno-second-underscore` to `FFLAGS` and even replace in `LIBS`, `-lgfortran` by `lg2c` (if version of mpiCC is 3.xx).

Both OpenMP and MPI

That could be as easy as typing:

```
../../utils/scons.py mpi=yes openmp=yes
```

But, the same difficulties encountered with the OpenMP and MPI buildings (see previous subsections) might pile up here ... with the same solutions.

The Good (?) Old Way

You don't want to bother with all this parallelizing stuff? No time? No thrill? Prefer old-shool style? But you still would like to shorten your aerosol simulation. Life is short after all! Whatever your reasons are, we did not forget you if you are running Polyphemus on a POSIX multi-core platform. Indeed, the aerosol modules `Aerosol_SIREAM_SORGAM` and `AerosolRACM_SIREAM_AEC` were also parallelized independantly from OpenMP and MPI using the good old `fork` and a shared memory segment. What are you required to do? Just compile your program normally and consult Section 6.3.2.

2.4 Editing your Configuration Files

Now that you are done with compiling and before you launch your own preprocessing or simulation jobs, you will have to edit and perhaps modify some configuration files to adapt them to your needs.

We thougt about it: next to each Polyphemus program, you will find at least one example of suitable configuration files. It shall be a good starting point as you could adapt this example to your own case with minor changes.

Let's take an example located in `preprocessing/meteo` to illustrate the configuration files of `meteo`. As you will see in 3.4.1, there are two of them that we will comment successively (in what follows, (n), where n is a number, refers to the comment written after the verbatim of the file). The first one, that defines the considered domain with its space and time discretization, is `general.cfg` and is located in `preprocessing`.

```
[general] (1)

Home: /u/cergrene/a/ahmed-dm (2)
Directory_computed_fields: <Home>/data
Directory_ground_data: <Directory_computed_fields>/ground
Programs: <Home>/src/polyphemus/core/trunk/preprocessing


[domain] (1)
```

```
Date: 2004-08-09_03-00 (3)
Delta_t = 3.0 (4)
x_min = -10.0 Delta_x = 0.5 Nx = 65
y_min = 40.5 Delta_y = 0.5 Ny = 33
Nz = 5
Vertical_levels: <Programs>/levels.dat (5)
```

(1) A configuration file is organized with sections like `[general]` and `[domain]`. Here, `[general]` is used to define paths that will be referred later on and `[domain]` contains the domain definition along with its space and time discretization.

(2) Within each section are defined fields like `Home`. `Home: /u/cergrene/a/ahmed-dm` indicates that the field `Home` takes the string `'/u/cergrene/a/ahmed-dm'` as value.

(3) Here is one possible date format with minutes : YYYY-MM-DD_HH-II (see Appendix D for further informations).

(4) `'Delta_t = 3.0'` indicates that the field `Delta_t` takes the floating point number 3.0 as value. We recommend use '=' for numerical values and ':' otherwise.

(5) `<Programs>` is a markup to the field `Programs`. It will be replaced by its actual value `/u/cergrene/a/ahmed-dm/src/polyphemus/core/trunk/preprocessing` when the program will read the configuration files.

The second one is `meteo.cfg` to be found in `preprocessing/meteo`:

```
[paths]

### Inputs. (1)

Database_meteo: /u/cergrene/B/quelo/Meteo_ECMWF/2001/

... (2)

# Roughness heights on the input ECMWF domain. Only needed if (1)
# 'Richardson_with_roughness' is set to true.
Roughness_in: <Directory_ground_data>/Roughness_in.bin (3)

### Outputs.

Directory_meteo: <Directory_computed_fields>/meteo/

... (2)

[ECMWF]

Date : &D (4)
t_min = 0. Delta_t = 3.0 Nt = 9
x_min = -15.12 Delta_x = 0.36 Nx = 168
y_min = 32.76 Delta_y = 0.36 Ny = 113
Nz = 31
```

```
[meteo]

# Should the surface Richardson number be computed taking into
# account roughness height?
Richardson_with_roughness: no (5)


[accumulated_data]

# For 'data' storing values cumulated in time.
# length number of time steps over which data is cumulated.
Accumulated_time = 4
# start (optional) index of the first complete cycle. Default: 0.
Accumulated_index = 1

... (2)
```

(1) In a line, everything that comes after '#' is a comment and is then ignored.

(2) '...' means that for readibility purpose, we removed here some useless fields or sections you will find in the actual file example. Indeed, because `meteo.cfg` serves also as the example configuration file of programs `attenuation` and `Kz`, fields and sections related to these programs are defined. But they will be discarded while running `meteo`.

(3) The markup `<Directory_ground_data>` is still available even if it was defined in `general.cfg`. Indeed, configuration files are concateneted before being read. Still, as markups cross sections you might be careful not to create ambiguous markups, that is markups that refer to a field name used in several sections!

(4) The field `Date` of the section `[ECMWF]` is not the same as the field `Date` of the section `[domain]` because the sections they belong to are distinct.

(5) 'no' is considered as a boolean. 'yes', true' and 'false' are also boolean supported by Polyphemus (see Appendix D for further details).

You might now be equipped to survive in the jungle of configuration files. For further informations about the configuration files, you are advised to have a look at the lexical reference given in the Appendix D. For informations about specific configuration files, you should read the section related to the program, the driver, the model or the module you would like to use.

2.5 Running the Programs

2.5.1 Running a Program from Command Line

Most programs require one or two input configuration files, and sometimes one or two dates (beginning and end dates, see Section 3.2.3). Most programs provide help when launched without any input file. Here is an example with the program `bio`[†1]:

[†1] Further details about specific programs are provided in chapter 3.

```
~/Polyphemus/preprocessing/bio/> ./bio

Usage:
  ./bio [main configuration file] [secondary config file] [first date] [second date/interval]
  ./bio [main configuration file] [first date] [second date/interval]
  ./bio [main configuration file] [secondary config file] [first date]
  ./bio [first date] [second date/interval]
  ./bio [first date]

Arguments:
  [main configuration file] (optional): main configuration file. Default: bio.cfg
  [secondary configuration file] (optional): secondary configuration file. Default: "".
  [first date]: beginning date in any valid format.
  [second date]: end date in any valid format.
  [interval] (optional): Interval in format NdMh or Nd-Mh or Nd or Mh where N is the
number of days and M the number of hours. Default: 1d.

Note:
  The end date, whether it is given directly or computed by adding the time interval to
the beginning date, is always considered as excluded.
```

Program `bio` takes from one to four arguments. Below are four possible calls:

```
./bio 2001-04-22
./bio bio.cfg 2001-04-22
./bio bio.cfg 2001-04-22 2001-04-23
./bio ../general.cfg bio.cfg 2001-04-22
```

The first three calls are equivalent. The fourth one involves two configuration files. The program `bio` behaves as if these two configuration files were merged. It means that the fields required by the program may be put in any of these two files. Markups defined in one file can be expanded in the other file. The only constraint is that each section should appear in a single file only.

2.5.2 Sharing Configuration

The command line:

```
./bio ../general.cfg bio.cfg 2001-04-22 1d
```

with the two configuration files `general.cfg` and `bio.cfg`, is the advocated line. The configuration file `general.cfg` gathers information that may be needed by several programs in the `preprocessing/` directory (`meteo`, `attenuation`, `luc-usgs`, etc.). Such a configuration file is provided with `Polyphemus/preprocessing/general.cfg`:

```
[general]

Home: /u/cergrene/0/bordas
Directory_computed_fields: <Home>/B/data
Directory_ground_data: <Directory_computed_fields>/ground
Programs: <Home>/codes/Polyphemus-HEAD
```

```
[domain]

Date: 2001-01-02_00-00-00
Delta_t = 3.0
x_min = -10.0   Delta_x = 0.5   Nx = 65
y_min = 40.5    Delta_y = 0.5   Ny = 33
Nz = 5
Vertical_levels: <Programs>/levels.dat
```

The simulation domain and the simulation date are defined. In addition, markups (`Directory_computed_fields`, `Directory_ground_data` and `Programs`) are introduced and may be referred by other configuration files such as `meteo.cfg`.

Actually most configuration files (`meteo.cfg`, `luc-usgs.cfg`, `emissions.cfg`, etc.) provided in Polyphemus, along with the programs, are examples that refer to the markups defined in `general.cfg`. Essentially three markups are defined in `general.cfg`:

- `Directory_computed_fields`: where output results (i.e., fields computed by preprocessing programs) are stored.
- `Directory_ground_data`: where ground data (land use cover, roughness) is stored.
- `Programs`: path to Polyphemus preprocessing directory.

Polyphemus configuration files are written so that mainly changes in `general.cfg` should be needed to perform a reference simulation. In `general.cfg`, one changes the paths (markups) to the preprocessing programs (`Programs`) and to the output results (`Directory_computed_fields` and `Directory_ground_data`), and one chooses its simulation domain. Other configuration files provide paths to input data (meteorological files, emissions data, etc.) and fine options.

2.5.3 Notes about the Models

To launch a simulation you have to compile and execute a C++ program, which is composed of a driver (on top of the model itself), a model and its modules (if any). See Section 1.1 for a short description of the flowchart. The program of the simulation looks like `processing/photochemistry/polair3d.cpp`: it is a short C++ code that declares the driver, the model and the modules.

You may have to modify this program in case you change the model, the driver or a module. In that case, duplicate `processing/photochemistry/polair3d.cpp` (or another example) and modify it according to the notes below. Actually it is likely that the model/driver combination is already in use in one of the examples: have a look in `processing/*/*.cpp`.

First determine which model you need, depending on your simulation target:

- for a passive simulation: `Polair3DTransport` or `CastorTransport`;
- for a simulation with chemistry for gaseous species: `Polair3DChemistry` or `CastorChemistry`;
- for a simulation with aerosol species: `Polair3DAerosol`;
- for a simulation with gaseous species and data assimilation: `Polair3DChemistryAssimConc`;
- for a simulation at local scale using an Eulerian model: `StationaryModel` with another Eulerian model as the underlying model (for instance `Polair3DChemistry`);

- for a simulation with a Gaussian plume model: `GaussianPlume`, or `GaussianPlume_aer` if there are aerosol species;
- for a non-stationary simulation at local scale with a Gaussian model: `GaussianPuff`, or `GaussianPuff_aer` if there are aerosol species;
- for a simulation with point sources, you can use the model `PlumeInGrid`, in order to improve the way the dispersion of the pollutants inside a cell is modelled;
- to simulate the dispersion of passive particles without deposition nor scavenging: `LagrangianTransport` with its related particle models.

To set the model, just modify the definition of `ClassModel`:

```
typedef MyModel<Argument(s)> ClassModel;
```

For instance:

```
typedef Polair3DAerosol<real, AdvectionDST3<real>,
    DiffusionROS2<real>, Decay<real> > ClassModel;
```

If you change a model, you may also change the modules (a model may need less modules or no module at all: remove them if necessary). The modules are (template) arguments of the model (`AdvectionDST3<real>`, `DiffusionROS2<real>` and `Decay<real>`, in the previous example). The order in which the modules are provided matters: it is always advection, diffusion and chemistry, or transport (single module which replaces advection and diffusion) and chemistry. See Section 2.6.5 for the modules you can use with the model you chose.

Then, in your main C++ program, declare the right driver. You may replace `BaseDriver` with a new driver at this line (in `processing/photochemistry/polair3d.cpp`):

```
BaseDriver<real, ClassModel, BaseOutputSaver<real, ClassModel> >
  Driver(argv[1]);
```

See Chapter 4 for the various drivers available and their use.

Finally make sure to include all models, modules, drivers and output savers you use (at the beginning of the file – statements `#include "...cxx"`). The `SConstruct` file or makefile may need changes too.

If you are not confident with your own changes, have a look at the examples: it is likely that you find a close combination there. In case you try an unusual combination, you may contact polyphemus-help@lists.gforge.inria.fr.

The directory named `processing` provides examples of configuration and data files to use with the programs. For instance, `processing/photochemistry` provides an example for a forward eulerian simulation combining advection-diffusion with chemistry. It should be launched *in* `processing/photochemistry`. Outputs will then be stored in the subdirectory `processing/photochemistry/results`, so make sure that this directory exists *before* you start the simulation (indeed Polyphemus programs *do not* create directories before saving results).

2.5.4 Running a parallelized program

Given you have compiled your program appropriately to make it parallel (see 2.3.3), let's see how to launch a parallel job.

The OpenMP way

The OpenMP way is without any doubt the easiest one. You can use it if you built your parallelized program "with OpenMP alone" (see 2.3.3) and if the field `Number_of_threads_openmp` is set to the desired value in the section `[computing]` of the main configuration file. For instance, in `processing/photochemistry/racm.cfg`:

```
[computing]

# Number of threads if openMP parallelization is used.
Number_of_threads_openmp: 4
```

Then, launching command for the related `processing/photochemistry/polair3d` remains unchanged. From `processing/photochemistry`, simply type:

```
polair3d racm.cfg
```

The MPI and OpenMP/MPI way

Whether you chose "MPI alone" or "both MPI and OpenMP" to build your parallel program, the procedure is the same. We will illustrate it supposing you have installed the Open MPI environment.

First, you should create a text file containing the hostname of your targeted machines. For example, let's call it `hostfile`:

```
node001
node002
node003
...
```

Then, you are almost done! Just launch your parallel job with the following command:

```
mpirun -np 8 --hostfile hostfile polair3d polair3d.cfg
```

where `np` indicates the number of nodes you want to use. If you are running your job on a single hexacore with MPI alone, `np` value has to be set to 6. In this case, there should be only one hostname in your `hostfile`. If you are running it on 10 different monocore-nodes, then you have to use MPI alone: the `hostfile` will contain 10 hostnames and `np` will be set to 10.

Using "both MPI and OpenMP" won't change the basics of the command line. For example, a job targeting 3 quadcore-nodes might be launched with:

```
mpirun -np 3 --hostfile hostfile polair3d polair3d.cfg
```

given `hostfile` indicates the 3 related nodes and the field `Number_of_threads_openmp` has been set to 4 in the configuration file `polair3d.cfg`.

Note 1: That was the Open MPI case but if you are still stuck with an old LAM/MPI environment, you will be asked for additional efforts. Finally, the following guidelines might convince you to migrate towards Open MPI!

Indeed, before launching the parallel job, you have got to create a "LAM universe". Don't panic: the "LAM universe" is only a set of processors which can interact using LAM/MPI commands. Let's have a look on the command that creates it but don't try it now:

```
lamboot -v -ssi boot_rsh_agent "ssh -A" hostfile
```

Some important remarks about this command:

- option `-ssi boot_rsh_agent "ssh -A"` is used to make sure that `ssh` and not `rsh` is used to connect to the other machines. You also have to use an ssh-agent in order to avoid being prompted for your passphrase when connecting. Indeed, any output from `ssh` would cause `lamboot` to fail. To check the connections with other nodes, it is a good idea to connect once to each machine "by hand" before using `lamboot`.

- Your "LAM universe" is described in a text file called `hostfile` which gives the name of the machines to use and the number of cores to use on each machine, for example:

```
node001 cpu=4
node002 cpu=4
node003 cpu=4
node004 cpu=4
node005 cpu=4
node006 cpu=4
...
```

Now, if your file `hostfile` is ready, you can type the `lamboot` command.

If your run ended with an error, it is advised to make sure that the system is clean (memory has been de-allocated, no-processes are still running, ...). To do so, launch the command:

```
lamclean
```

If you want to shutdown the "LAM universe" , type:

```
lamhalt
```

In case `lamclean` has failed, you can use `lamwipe` instead which cleans the environment and closes it. You are advised though to use `lamclean` then `lamhalt`.

At this point, we hope you managed to create your "LAM universe". It is time to show you how to launch a parallel job. We take the example of a parallelized version of `processing/photochemistry/polair3d`. To launch

the run, type:

```
mpiexec -v -np 14 polair3d racm.cfg
```

Some remarks about the command above:

- the option `-np 14` lets launch the program with 14 nodes (automatically selected). If you want to use a given number of processes per node, you can use option `N` instead and for one process per CPU, use `C`. If you want to launch the program on nodes 0 to 7 for instance, put option `n0-7`.

- The option `-v` makes `mpiexec` verbose, which can be useful in order to know on what nodes the program was launched.

Note 2: In Unix-like systems, the command `ipcs` can be helpful to check whether `lamclean` was entirely successfull. If, for any reason, your program crashed and despite `lamclean`, some orphan semaphors are left, you might kill them thanks to `ipcrm`.

Note 3: In the case of an MPI/OpenMP-built program, the field `Number_of_threads_openmp` has to be propely set in the main configuration file as indicated above in the subsection dedicated to the OpenMP way.

The good (?) old way

If you followed in 2.3.3 the "good old way" to parallelize one of the aerosol modules `Aerosol_SIREAM_SORGAM` or `Aerosol_SIREAM_AEC`, please consult directly Section 6.3.2). The launching command does not differ from the one of the serial program. You will just have to set the appropriate field in the main configuration file.

2.6 Setting Up a Simulation

This section is a quick overview of how a simulation should be set up. It is not meant to and cannot replace the chapters about preprocessing, models, modules, ...

2.6.1 Suggested Directory Tree

It is advocated not to modify Polyphemus code, including the configuration files provided with it. The whole Polyphemus directory should not be modified (except maybe `SConstruct` files or makefiles). Copy the configuration files you need in a dedicated directory, modify the new configuration files in this directory, and run Polyphemus programs from this directory. Your directory tree may look like:

```
Polyphemus-{version}/include/
                    /postprocessing/
                    /preprocessing/
                    /processing/
                    /utils/
MyStudy/config/
       /data/emissions/
            /meteo/
            /[...]
       /raw_data/
       /results/reference/
               /new_emissions/
               /[...]
```

where `MyStudy` contains Polyphemus configurations files set for the study (`configuration` with `general.cfg`, `meteo.cfg`, ... in it), data generated by preprocessing programs (directory `data`), sometimes raw data (directory `raw_data`) necessary for preprocessing and finally output results from the simulation (directory `results`, with results from different runs).

Notice that Polyphemus directory includes the version number (or the date). This is very useful in order to properly track simulations. In directory `MyStudy`, you should add a file called `version` which should contain Polyphemus version (and maybe the version of other tools).

You may also want to copy configuration files in your output directory. For instance, you may copy `meteo.cfg` in directory `MyStudy/data/meteo/` so as to know with which configuration your meteorological data were generated.

2.6.2 Roadmaps

Roadmaps with "Polair3D" Models

In short, the main steps to set up an Eulerian simulation with model `Polair3D` are:

1. generation of ground data (land use cover, roughness height) – `preprocessing/ground`;
2. preprocessing of meteorological fields – `preprocessing/meteo`;
3. other preprocessing steps if relevant (deposition velocities, emissions, ...);
4. compiling the right combination of model, module(s) and driver (see Sections 2.6.5 and 2.5.3).

Passive tracer Below is a possible sequence of programs to be launched to perform a basic passive simulation:

```
preprocessing/ground/luc-glcf
preprocessing/ground/roughness
preprocessing/meteo/MM5-meteo
preprocessing/meteo/Kz_TM
processing/transport/polair3d-transport
```

Program `polair3d-transport` is not provided with Polyphemus (subdirectory `processing/transport` should also be created). It should be built with Polyphemus components: `BaseDriver` (driver), `Polair3DTransport` (model), `AdvectionDST3` (module), `DiffusionROS2` (module). See Section 2.5.3 for details.

Photochemistry Below is a possible sequence of programs to be launched to perform a photochemistry simulation:

```
preprocessing/ground/luc-glcf
preprocessing/ground/roughness
preprocessing/meteo/MM5-meteo
preprocessing/meteo/Kz_TM
preprocessing/emissions/emissions
preprocessing/bio/bio
preprocessing/dep/dep
preprocessing/ic/ic
preprocessing/bc/bc
processing/photochemistry/polair3d
```

Aerosol Below is a possible sequence of programs to be launched to perform a simulation with aerosol species:

```
preprocessing/ground/luc-glcf
preprocessing/ground/roughness
preprocessing/ground/luc-convert
preprocessing/meteo/MM5-meteo
preprocessing/meteo/Kz_TM
preprocessing/emissions/emissions
preprocessing/emissions/sea_salt
```

```
preprocessing/bio/bio
preprocessing/dep/dep
preprocessing/ic/ic
preprocessing/bc/bc
preprocessing/bc/bc-gocart (4 times in a row)
preprocessing/bc/bc-nh4
processing/siream-sorgam/polair3d-siream-racm
```

Roadmap with "Castor" Models

The roadmap with "Castor" models is very similar to the one with "Polair3D" models except that raw data and preprocessing programs used to modify them are often different.

Below is a possible sequence of programs to be launched to perform a photochemistry simulation with Castor model:

```
preprocessing/ground/ground-castor.py
preprocessing/meteo/MM5-meteo-castor
preprocessing/emissions/chimere_to_castor
preprocessing/bio/bio-castor
preprocessing/dep/dep-emberson
preprocessing/ic/ic
preprocessing/bc/bc-inca
processing/castor/castor
```

Roadmaps with Gaussian Models

In short, the main steps to set up a Gaussian simulation are:

1. generation of meteorological data: no program is available to do it, but as only little information is required this should be quite easy. Examples of meteorological files are provided in `processing/gaussian/gaussian-meteo.dat` and `processing/gaussian/gaussian-meteo_aer.dat`.

2. preprocessing: `discretization` to generate source files for line emission and `gaussian-deposition` or `gaussian-deposition_aer` to compute deposition velocities and scavenging coefficients(without or with aerosol species respectively). For more details, see Section 3.9.

3. compiling the right combination of model (`GaussianPlume`, `GaussianPlume_aer`, `GaussianPuff`, `GaussianPuff_aer`) and driver (`PlumeDriver` or `PuffDriver`).

2.6.3 Mandatory Data in Preprocessing

ECMWF Fields

In ECMWF files, it is recommended to have the following fields (with their Grib codes):

- Volumetric soil water layer 1 (39),
- Volumetric soil water layer 2 (40),
- Volumetric soil water layer 3 (41),

- Volumetric soil water layer 4 (42),
- Temperature [3D] (130),
- U velocity [3D] (131),
- V velocity [3D] (132),
- Specific humidity [3D] (133),
- Snow depth (141),
- Stratiform precipitation (Large-scale precipitation) [accumulated] (142),
- Convective precipitation [accumulated] (143),
- Snowfall (convective + stratiform) [accumulated] (144),
- Surface sensible heat flux [accumulated] (146),
- Surface latent heat flux [accumulated] (147),
- Logarithm of surface pressure (152),
- Boundary layer height (159),
- Total cloud cover (164),
- 2 meter temperature (167),
- Surface solar radiation downwards [accumulated] (169),
- Surface solar radiation [accumulated] (176),
- East-West surface stress [accumulated] (180),
- North-South surface stress [accumulated] (181),
- Evaporation [accumulated] (182),
- Low cloud cover (186),
- Medium cloud cover (187),
- High cloud cover (188),
- Skin temperature (235),
- Forecast albedo (243),
- Cloud liquid water content [3D] (246),
- Cloud ice water content [3D] (247),
- Cloud cover [3D] (248).

Not all data may be required, depending on the programs you actually run.

2.6.4 Mandatory Data for Models

Table 2.3 presents all variables needed by various models (and the name under which they appear in the data configuration files). Note that additional data can be necessary to add initial conditions, boundary conditions, source terms (volume emissions, surface emissions) or loss terms (deposition velocities, scavenging).

Table 2.3: Mandatory data for each models.

Model	Data necessary.
CastorTransport	Temperature, Pressure, Altitude, AirDensity, MeridionalWind ZonalWind VerticalDiffusion.
CastorChemistry	Same as CastorTransport and SpecificHumidity, LiquidWaterContent, Attenuation.
Polair3DTransport	MeridionalWind (for advection), ZonalWind (for advection), VerticalDiffusion (for diffusion), Horizontal_diffusion (if `Isotropic_diffusion` is set to no; this value is given in the main configuration file), Temperature (if `With_air_density` is set to yes or for microphysical scavenging model), Pressure (if `With_air_density` is set to yes or for microphysical scavenging model).
Polair3DChemistry	Same as Polair3DTransport and SpecificHumidity, Attenuation.
Polair3DChemistryAssimConc	Same as Polair3DChemistry.
Polair3DAerosol	Same as Polair3DChemistry and LiquidWaterContent, SnowHeight.
PlumeInGrid	Same as Polair3DChemistry and LowCloudiness, MediumCloudiness, HighCloudiness, SolarRadiation, FirstLevelWindModule, FrictionModule, BoundaryHeight, LMO (Monin-Obukhov length).
GaussianPlume or GaussianPlume_aer	Temperature, Wind_angle, Wind (wind module),

	Boundary_height, Stability (if `Briggs` or `Doury` are used), Friction_velocity (if `similarity_theory` is used), Convective_velocity (if `similarity_theory` is used), LMO (if `similarity_theory` is used), Coriolis (Coriolis parameter) (if `similarity_theory` is used).
GaussianPuff or GaussianPuff_aer	Same as Gaussian and Attenuation (if `with_chemistry` is set to yes), Pressure (if `with_chemistry` is set to yes), Specific_humidity (if `with_chemistry` is set to yes).
LagrangianTransport	MeridionalWind, ZonalWind, VerticalDiffusion Horizontal_diffusion (not always mandatory depending on the model of particle; this value is given in the main configuration file), Temperature, Pressure.

For Gaussian models, the data are single values read by the models in a configuration file. All data for Eulerian and Lagrangian models are 4D-fields, outputs of meteorological preprocessing programs:

- `meteo`, `Kz`, `attenuation` (and `Kz_TM` if you use Troen & Mahrt parameterization for vertical diffusion), for models of type "Polair3D" or "LagrangianTransport" while using raw meteorological data from ECMWF.
- `MM5-meteo` (and `Kz_TM` if you use Troen & Mahrt parameterization for vertical diffusion) for models of type "Polair3D" or "LagrangianTransport" while using raw meteorological data from model MM5;
- `MM5-meteo-castor` (and `Kz_TM` if you use Troen & Mahrt parameterization for vertical diffusion) for models of type "Castor" while using raw meteorological data from model MM5.

Remember that in addition data from programs `emissions`, `dep`, `ic`, ..., can be needed.

2.6.5 Models / Modules Compatibilities

Models of type "Polair3D" require two transport modules (one for advection and one for diffusion), while models of type "Castor" only require one transport module (which deals with advection and diffusion). This does not mean that a module could not be shared by both models (although there is no common module in current Polyphemus version).

Table 2.4 and Table 2.5 present a summary of the compatibility between models and modules. Note that Gaussian models are not included in these tables because they don't need any module.

In Table 2.4, `Polair3DAssim`* designates the assimilation model `Polair3DChemistryAssimConc`.

In Table 2.5, module names are shortened to be displayed on one line: `Castor` is actually `ChemistryCastor`, `PhotoChem` is `Photochemistry`, `RADM` is `ChemistryRADM`, `SORGAM` is `Aerosol_SIREAM_SORGAM` and `AEC` is `Aerosol_SIREAM_AEC`.

Table 2.4: Compatibility between models and transport modules.

	AdvectionDST3	DiffusionROS2	TransportPPM
Polair3DTransport	X	X	
Polair3DChemistry	X	X	
Polair3DAerosol	X	X	
Polair3DAssim*	X	X	
PlumeInGrid	X	X	
CastorTransport			X
CastorChemistry			X

	SplitAdvectionDST3	GlobalAdvectionDST3	GlobalDiffusionROS2
Polair3DTransport	X	X	X
Polair3DChemistry	X	X	X
Polair3DAerosol	X	X	X
Polair3DAssim*	X	X	X
PlumeInGrid	X	X	X
CastorTransport			
CastorChemistry			

Table 2.5: Compatibility between models and chemistry modules.

	Castor	PhotoChem	RADM	SORGAM	AEC	Decay
Polair3DChemistry		X	X	X	X	X
Polair3DAerosol				X	X	X
Polair3DAssim*		X	X			
PlumeInGrid		X				
CastorChemistry	X					

Please note that 4DVar assimilation scheme only works with RACM chemistry module.

As for drivers, BaseDriver is the simplest and the most used of them. The other drivers available are:

- `PlumeDriver`: for Gaussian plume model (with or without aerosol species).
- `PuffDriver`: for Gaussian puff model (with or without aerosol species).
- Data assimilation drivers: `OptimalInterpolationDriver` (optimal interpolation), `EnKFDriver` (ensemble Kalman filter), `RRSQRTDriver` (reduced rank square root filter), `FourDimVarDriver` (4D-Var), to be associated with `Polair3DAssimConc` model.
- Drivers for the verification of adjoint coding in variational assimilation: `AdjointDriver`, `GradientDriver`, `Gradient4DVarDriver`.

2.7 Checking Results

It is highly recommended to check the fields generated by Polyphemus programs: meteorological fields, deposition velocities, output concentrations, ...

First, you should check the size of the binary files. Be it with preprocessing or processing programs, results are saved as floating point numbers with single precision. As we will express file size in bytes, a factor 4 will appear below.

2.7.1 Checking the output file size of preprocessing programs

In what follows:

- take N_z, N_y, N_x and N_t from the section `[domain]` of the configuration files;
- N_{days} is the number of preprocessed days and
- N_{total} is the total number of preprocessing time steps: $N_{total} = N_t \times N_{days}$

Here are some indications on what to check depending on the program.

Ground Data

As there are 14 categories in the GLCF description, the size of the binary file obtained from `luc-glcf` should be $N_x \times N_y \times 4 \times 14$. As of `luc-usgs`, it should be $N_x \times N_y \times 4 \times 24$.

Meteorological fields

For programs `meteo`, `attenuation`, `Kz`, `MM5-meteo`, `WRF-meteo` and `Kz_TM`, related file size :

- for 2D fields is $N_{total} \times N_y \times N_x \times 4$;
- for 3D fields is $N_{total} \times N_z \times N_y \times N_x \times 4$;
- except for the fields `Kz` and `Kz_TM`, $N_{total} \times (N_z + 1) \times N_y \times N_x \times 4$;
- for the field `MeridionalWind`, $N_{total} \times N_z \times (N_y + 1) \times N_x \times 4$;
- and for the field `ZonalWind`, $N_{total} \times N_z \times N_y \times (N_x + 1) \times 4$.

Deposition Velocities

For program `dep`, the size of output files should be $N_{total} \times N_y \times N_x \times 4$.

Anthropogenic Emissions (EMEP)

For program `emissions`, the time step is of 1 hour so there are 24 time steps per day. Related file size for surface emissions will then be of $N_{days} \times 24 \times N_y \times N_x \times 4$.

Things are a little bit trickier for volume emissions as we do not know the number of vertical levels where emissions occur. This number could be lower than N_z as, for instance, there are no EMEP emissions above 1106 meters. The only thing we know is then that the file size should be a multiple of $N_{days} \times 24 \times N_y \times N_x \times 4$.

Biogenic Emissions

The time step for biogenic emissions is not the one given in the section `[domain]` of `general.cfg` but the one specified in the section [biogenic] of `bio.cfg`. 1 hour is a popular value for it. In this case, the size of `bio` output files should be $24 \times N_{days} \times N_y \times N_x \times 4$.

Initial Conditions

Initial conditions generated by program `ic` should produce files of $N_z \times N_y \times N_x \times 4$ bytes.

Boundary conditions

To compute boundary conditions, programs `bc` and `bc-dates` use Mozart files that generally cover a period of 10 days with a timestep of 24 hours. Take notice that this is not the case for the beginning of January whose Mozart file provides data for the first 5 days of the month. As `bc` and `bc-dates` generate boundary conditions for the whole period covered by the Mozart files they use, regardless of the period that is actually of interest for your simulation, you should figure out how many days they indeed preprocess. It will give you the N_{days} to use.

For example, the command

```
./bc ../general.cfg bc.cfg h0067.nc
```

preprocesses one Mozart file. Then, $N_{days} = 10$.

But with the command

```
./bc-dates ../general.cfg bc-dates.cfg 2004-07-30 2004-08-12
```

you will deal with the files hc0061.nc and hc0062.nc (see 3.7 for details) that is two 10-days Mozart files. Then, $N_{days} = 20$.

Finally, the size of output files for boundary conditions along:

- z, should be $N_{days} \times N_y \times N_x \times 4$,
- y, should be $N_{days} \times N_z \times N_x \times 2 \times 4$,
- x, should be $N_{days} \times N_z \times N_y \times 2 \times 4$.

2.7.2 Checking the output file size of processing programs

If you set up to save the concentrations for each vertical level and at each time step, results file should be of size $4 \times N_t \times N_z \times N_y \times N_x$ bytes where N_x and N_y are the space steps along x and y directions respectively, N_z is the number of vertical levels of the field and N_t is the number of time steps, all of them specified in the main configuration file.

But if you specified the levels and a time interval on which to save, you might have to use the above formula with the suitable values of N_z and N_t. Indeed, they could be different from the values N_z and N_t specified in the main configuration file.

2.7.3 Checking the values

You should also check that the fields have reasonable values using the programs from directory `utils`, mainly `get_info_float` (see Section 2.8.1). The command line to use `get_info_float` is:

```
get_info_float results/O3.bin
```

And the output looks like:

```
Minimum: 0.0563521
Maximum: 169.219
Mean:  91.3722
```

2.8 Useful Tools

A few useful tools are provided in directory `Polyphemus/utils/`. Here is a brief explanation of their aim and their use.

2.8.1 Information about Binary Files

Two programs provided in `Polyphemus/utils/` are meant to provide information about the content of one or several binary files. It is highly recommended to use these programs to check the output files of preprocessing programs and drivers/models (e.g. in Section 2.7).

These two programs perform the same thing but on binary files with different floating precision:

- `get_info_float` gives the minimum, maximum and mean of binary files in single precision.
- `get_info_double` gives the minimum, maximum and mean of binary files in double precision.

It is assumed that the binary file to be analyzed by `get_info_float` or `get_info_double` contains only floating point numbers. No extra data such as headers should be in the file. Output binary files from preprocessing programs and from drivers/models satisfy this condition and can be properly read by `get_info_float` or `get_info_double`. Note that Polyphemus programs usually generate single precision files: it is very likely that one only uses `get_info_float`.

Using `get_info_float` or `get_info_double` is straightforward:

```
$ get_info_float Temperature.bin

Minimum: 257.621
Maximum: 300.882
Mean: 282.262

$ get_info_float Temperature.bin Pressure.bin

-- File "Temperature.bin"
Minimum: 257.621
Maximum: 300.882
Mean: 282.262

-- File "Pressure.bin"
Minimum: 56369.2
Maximum: 102496
Mean: 87544.1
```

2.8.2 Differences between Two Binary Files

There are two different types of programs to compute statistics about the differences between two files:

- `get_diff_precision` (where `precision` is `float` or `double`) returns statistics about the difference between two files. The files should only contain floating point numbers without headers.

- `get_partial_diff_precision` (where `precision` is `float` or `double`) returns statistics about the difference between two files. If these two files have the same size, `get_partial_diff_precision` does the same as `get_diff_precision`. If the files do not have the same size, only the first values (as much as possible) are compared.

Here is an example with `get_diff_float` launched from `processing/photochemistry/results`:

```
../../../utils/get_diff_float O3-ref.bin O3-diff.bin

                File #0         File #1
Minima:         0.0145559       0.0181665
Maxima:         136.795         175.123
Means:          71.578          65.4088
Standard dev.:  26.958          28.643

                Difference
Minimum:        -57.324
Maximum:        66.9219
Mean:           6.16919
Standard dev.:  14.4999

Correlation between files #0 and #1: 0.865696
```

2.8.3 MM5 Files

It can be useful to get information about MM5 files, in particular to modify the configuration file `MM5-meteo.cfg` (see Section 3.4.5). To do so, two programs are provided:

- `MM5_var_list` gives a list of all variables stored in a MM5 file. It also gives miscellaneous information about the file. Information provided can be needed in preprocessing step (program `MM5-meteo` – Section 3.4.5): number of space steps, time step and projection type.
- `get_info_MM5` gives the minimum, maximum, mean and standard deviation of a variable stored in a MM5 file (use program `MM5_var_list` to know what variables are stored in the file).

For instance, the output of `MM5_var_list` for the file `MM5-2004-08-09` used in Polair3D test case (see Appendix A) is:

```
Metadata (-999 means unknown):

OUTPUT FROM PROGRAM MM5 V3 : 11
TERRAIN VERSION 3 MM5 SYSTEM FORMAT EDITION NUMBER : 1
TERRAIN PROGRAM VERSION NUMBER : 6
TERRAIN PROGRAM MINOR REVISION NUMBER : 0
COARSE DOMAIN GRID DIMENSION IN I (N-S) DIRECTION : 76
COARSE DOMAIN GRID DIMENSION IN J (E-W) DIRECTION : 86
MAP PROJECTION. 1: LAMBERT CONFORMAL, 2: POLAR STEREOGRAPHIC, 3: MERCATOR : 1
IS COARSE DOMAIN EXPANDED?, 1: YES, 0: NO : 0
EXPANDED COARSE DOMAIN GRID DIMENSION IN I DIRECTION : 76
EXPANDED COARSE DOMAIN GRID DIMENSION IN J DIRECTION : 86
GRID OFFSET IN I DIR DUE TO COARSE GRID EXPANSION : 0
GRID OFFSET IN J DIR DUE TO COARSE GRID EXPANSION : 0
DOMAIN ID : 1
MOTHER DOMAIN ID : 1
NEST LEVEL (0: COARSE MESH) : 0
DOMAIN GRID DIMENSION IN I DIRECTION : 76
DOMAIN GRID DIMENSION IN J DIRECTION : 86
I LOCATION IN THE MOTHER DOMAIN OF THE DOMAIN POINT (1,1) : 1
J LOCATION IN THE MOTHER DOMAIN OF THE DOMAIN POINT (1,1) : 1
DOMAIN GRID SIZE RATIO WITH RESPECT TO COARSE DOMAIN : 1
                                                     : 1
REGRID Version 3 MM5 System Format Edition Number : 2
REGRID Program Version Number : 16
REGRID Program Minor Revision Number : 1
COARSE DOMAIN GRID DISTANCE (m) : 36000
COARSE DOMAIN CENTER LATITUDE (degree) : 47
COARSE DOMAIN CENTER LONGITUDE (degree) : 6
CONE FACTOR : 0.715567
TRUE LATITUDE 1 (degree) : 60
TRUE LATITUDE 2 (degree) : 30
POLE POSITION IN DEGREE LATITUDE : 90
APPROX EXPANSION (m) : 360000
GRID DISTANCE (m) OF THIS DOMAIN : 36000
I LOCATION IN THE COARSE DOMAIN OF THE DOMAIN POINT (1,1) : 1
J LOCATION IN THE COARSE DOMAIN OF THE DOMAIN POINT (1,1) : 1
I LOCATION IN THE COARSE DOMAIN OF THE DOMAIN POINT (IX,JX) : 76
J LOCATION IN THE COARSE DOMAIN OF THE DOMAIN POINT (IX,JX) : 86
TERRAIN DATA RESOLUTION (in degree) : 0.0833333
LANDUSE DATA RESOLUTION (in degree) : 0.0833333

MM5 Version 3 MM5 System Format Edition Number : 1
```

```
MM5 Program Version Number : 6
MM5 Program Minor Revision Number : 1
FOUR-DIGIT YEAR OF START TIME : 2004
INTEGER MONTH OF START TIME : 8
DAY OF THE MONTH OF THE START TIME : 9
HOUR OF THE START TIME : 0
MINUTES OF THE START TIME : 0
SECONDS OF THE START TIME : 0
TEN THOUSANDTHS OF A SECOND OF THE START TIME : 0
MKX: NUMBER OF LAYERS IN MM5 OUTPUT : 25
TIMAX:   SIMULATION END TIME (MINUTES) : 5760
TISTEP:  COARSE-DOMAIN TIME STEP IN SECONDS : 100
TAPFRQ:  TIME INTERVAL (MINUTES) THAT DATA WERE SAVED FOR GRIN : 60

Outputs:

Name      Dim. 1   2   3   4   Stag. Ord. Units            Description
U         3    76  86  25      D     YXS  m/s              U COMPONENT OF
                                                           HORIZONTAL WIND
V         3    76  86  25      D     YXS  m/s              V COMPONENT OF
                                                           HORIZONTAL WIND
T         3    76  86  25      C     YXS  K                TEMPERATURE
Q         3    76  86  25      C     YXS  kg/kg            MIXING RATIO
CLW       3    76  86  25      C     YXS  kg/kg            CLOUD WATER MIXING RATIO
RNW       3    76  86  25      C     YXS  kg/kg            RAIN WATER MIXING RATIO
ICE       3    76  86  25      C     YXS  kg/kg            CLOUD ICE MIXING RATIO
SNOW      3    76  86  25      C     YXS  kg/kg            SNOW MIXING RATIO
GRAUPEL   3    76  86  25      C     YXS  kg/kg            GRAUPEL MIXING RATIO
RAD TEND  3    76  86  25      C     YXS  K/DAY            ATMOSPHERIC RADIATION TENDENCY
W         3    76  86  26      C     YXW  m/s              VERTICAL WIND COMPONENT
PP        3    76  86  25      C     YXS  Pa               PRESSURE PERTURBATION
PSTARCRS  2    76  86          C     YX   Pa               (REFERENCE) SURFACE PRESSURE
                                                           MINUS PTOP
GROUND T  2    76  86          C     YX   K                GROUND TEMPERATURE
RAIN CON  2    76  86          C     YX   cm               ACCUMULATED CONVECTIVE
                                                           PRECIPITATION
RAIN NON  2    76  86          C     YX   cm               ACCUMULATED NONCONVECTIVE
                                                           PRECIPITATION
TERRAIN   2    76  86          C     YX   m                TERRAIN ELEVATION
MAPFACCR  2    76  86          C     YX   (DIMENSIONLESS)  MAP SCALE FACTOR
MAPFACDT  2    76  86          D     YX   (DIMENSIONLESS)  MAP SCALE FACTOR
CORIOLIS  2    76  86          D     YX   1/s              CORIOLIS PARAMETER
RES TEMP  2    76  86          C     YX   K                INFINITE RESERVOIR SLAB
                                                           TEMPERATURE
LATITCRS  2    76  86          C     YX   DEGREES          LATITUDE (SOUTH NEGATIVE)
LONGICRS  2    76  86          C     YX   DEGREES          LONGITUDE (WEST NEGATIVE)
LAND USE  2    76  86          C     YX   category         LANDUSE CATEGORY
TSEASFC   2    76  86          C     YX   K                SEA SURFACE TEMPERATURE
PBL HGT   2    76  86          C     YX   m                PBL HEIGHT
REGIME    2    76  86          C     YX   (DIMENSIONLESS)  PBL REGIME
SHFLUX    2    76  86          C     YX   W/m^2            SENSIBLE HEAT FLUX
LHFLUX    2    76  86          C     YX   W/m^2            LATENT HEAT FLUX
UST       2    76  86          C     YX   m/s              FRICTIONAL VELOCITY
SWDOWN    2    76  86          C     YX   W/m^2            SURFACE DOWNWARD SHORTWAVE
                                                           RADIATION
LWDOWN    2    76  86          C     YX   W/m^2            SURFACE DOWNWARD LONGWAVE
                                                           RADIATION
SWOUT     2    76  86          C     YX   W/m^2            TOP OUTGOING SHORTWAVE
                                                           RADIATION
LWOUT     2    76  86          C     YX   W/m^2            TOP OUTGOING LONGWAVE
```

```
                                                                    RADIATION
SOIL T 1  2    76  86          C     YX   K                 SOIL TEMPERATURE IN LAYER  1
SOIL T 2  2    76  86          C     YX   K                 SOIL TEMPERATURE IN LAYER  2
SOIL T 3  2    76  86          C     YX   K                 SOIL TEMPERATURE IN LAYER  3
SOIL T 4  2    76  86          C     YX   K                 SOIL TEMPERATURE IN LAYER  4
SOIL M 1  2    76  86          C     YX   m^3/m^3          TOTAL SOIL MOIS IN LYR  1  4
SOIL M 2  2    76  86          C     YX   m^3/m^3          TOTAL SOIL MOIS IN LYR  2  4
SOIL M 3  2    76  86          C     YX   m^3/m^3          TOTAL SOIL MOIS IN LYR  3  4
SOIL M 4  2    76  86          C     YX   m^3/m^3          TOTAL SOIL MOIS IN LYR  4  4
SOIL W 1  2    76  86          C     YX   m^3/m^3          SOIL LQD WATER IN LYR  1   4
SOIL W 2  2    76  86          C     YX   m^3/m^3          SOIL LQD WATER IN LYR  2   4
SOIL W 3  2    76  86          C     YX   m^3/m^3          SOIL LQD WATER IN LYR  3   4
SOIL W 4  2    76  86          C     YX   m^3/m^3          SOIL LQD WATER IN LYR  4   4
CANOPYM   2    76  86          C     YX   m                 CANOPY MOISTUR E CONTENT
WEASD     2    76  86          C     YX   mm                WATER EQUIVALENT SNOW DEPTH
SNOWH     2    76  86          C     YX   m                 PHYSICAL SNOW DEPTH
SNOWCOVR  2    76  86          C     YX   fraction          FRACTIONAL SNOW COVER
ALB       2    76  86          C     YX   fraction          ALBEDO
GRNFLX    2    76  86          C     YX   W m{-2}           GROUND HEAT FLUX
VEGFRC    2    76  86          C     YX   fraction          VEGETATION COVERAGE
SEAICE    2    76  86          C     YX   (DIMENSIONLESS)   SEA ICE FLAG
SFCRNOFF  2    76  86          C     YX   mm                SURFACE RUNOFF
UGDRNOFF  2    76  86          C     YX   mm                UNDERGROUND RUNOFF
T2        2    76  86          C     YX   K                 2-meter Temperature
Q2        2    76  86          C     YX   kg kg{-1}         2-meter Mixing Ratio
U10       2    76  86          C     YX   m s{-1}           10-meter U Component
V10       2    76  86          C     YX   m s{-1}           10-meter V Component
ALBD      2    27  2                CA   PERCENT           SURFACE ALBEDO
SLMO      2    27  2                CA   fraction          SURFACE MOISTURE AVAILABILITY
SFEM      2    27  2                CA   fraction          SURFACE EMISSIVITY AT 9 um
SFZ0      2    27  2                CA   cm                SURFACE ROUGHNESS LENGTH
THERIN    2    27  2                CA   100*cal cm^-2
                                           K^-1 s^1/2        SURFACE THERMAL INERTIA
SFHC      2    27  2                CA   J m^-3 K^-1       SOIL HEAT CAPACITY
SCFX      1    27                   CA   fraction          SNOW COVER EFFECT
SIGMAH    1    25              H    S    sigma             VERTICAL COORDINATE

Total number of time steps read in the file: 97
```

For each variable is provided:

- its name;
- its number of dimensions;
- its length along dimension 1 (if applicable);
- its length along dimension 2 (if applicable);
- its length along dimension 3 (if applicable);
- its length along dimension 4 (if applicable);
- the position at which the variable is given (`Stag.`): dot points (`D`, corner of the grid squares) or cross points (`C`, center of the grid squares);
- its dimensions ordering;
- its unit (or `(DIMENSIONLESS)`);

- a short description.

Then you can use the program `get_info_MM5` to have statistical data about one of the variables. Note that some variables have a blank space in their name so in that case you need to put the name between quotes to use `get_info_MM5`. If the name has no blank spaces, quotes are not necessary but can be used.

```
~/TestCase/raw_data/MM5> get_info_MM5 MM5-2004-08-09 'GROUND T'

Min: 271.911
Max: 327.747
Mean: 294.112
Std. dev.: 6.46779

~/TestCase/raw_data/MM5> get_info_MM5 MM5-2004-08-09 ALB

Min: 0.0738
Max: 0.8
Mean: 0.122658
Std. dev.: 0.0501975
```

Note that `get_info_MM5` only gives information on *one* field stored in the MM5 file.

2.8.4 Script `call_dates`

The script `call_dates` allows to call a program (in particular for preprocessing) over several consecutive days. Launch it without arguments to get help:

```
~/Polyphemus/utils> ./call_dates

Script "call_dates" calls a program over a range of dates.
Usage:
  "call_dates" [program] {arguments} [first date] [second date / number of days]
Arguments:
  [program]: program to be launched over the range of dates.
  {arguments}: arguments. Any occurence of %D is replaced with the date;
               otherwise the date is assumed to be the last argument.
  [first date]: first date of the range of dates.
  [second date / number of days]: last date of the range of dates
                                  or number of days of this range.
```

Below is an example:

```
~/Polyphemus/utils> call_dates echo "Current date:" 20060720 20060722

--------------------------------------------------------------------------------
nice time echo Current date: 20060720
Current date: 20060720
0.00user 0.00system 0:00.00elapsed 0%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+176minor)pagefaults 0swaps
```

```
--------------------------------------------------------------------------------
nice time echo Current date: 20060721
Current date: 20060721
0.00user 0.00system 0:00.00elapsed 0%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+177minor)pagefaults 0swaps
--------------------------------------------------------------------------------
nice time echo Current date: 20060722
Current date: 20060722
0.00user 0.00system 0:00.00elapsed 200%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+177minor)pagefaults 0swaps
--------------------------------------------------------------------------------
```

For each day, the command that is launched is shown (note that `nice time` has been prepended) and its output is displayed below.

Please note that `call_dates` can only process dates given in the format YYYYMMDD and not any of the other formats given in the Section D.7.

2.8.5 Other Utilities

The program `check_observation` checks that the content of an observation data file is consistent with Polyphemus conventions. This program can identify several problems in the data files. Launch `check_observation -h` to get help.

Many other utilities to be compiled with `scons.py` or `make` are provided to manipulate binary and text files. Below are short descriptions of these utilities:

- `add_float`: adds two single-precision binary files;
- `add_nb_float`: adds a number to a single-precision binary file;
- `double_to_float`: converts a double-precision binary file to a single-precision binary file;
- `float_to_text`: converts a single-precision binary file to a text file;
- `mult_nb_float`: multiplies a single-precision binary file by a given number;
- `reverse`: switch from big-endian to little-endian, or vice versa;
- `subtraction_float`: subtracts two single-precision binary files;
- `text_to_float`: converts a text file to a single-precision binary file.

To get further help on a program, launch this program without arguments. It will print help on screen.

Three Python scripts are provided for convenience:

- `replace_string` which performs string replacement in one or more files;
- `apply_on_files` which applies a given command to a list of files;
- `format` which formats a source code to meet certain requirements of Polyphemus coding standards.

Launch these programs with option `-h` to get help.

2.9 Ensemble Generation

In directory `utils/ensemble_generation/`, the Python module `ensemble_generation` is provided. It helps generating ensembles of any size.

2.9.1 Requirements

About `dsh` `dsh` is an implementation of a wrapper for executing multiple remote shell commands. Further information about this tool is available on the web site http://www.netfort.gr.jp/~dancer/software/dsh.html.en. You have to create a directory `$HOME/.dsh/group` and put a file in it with the list of hosts. Each line of this file has to contain a host name.

To check that `dsh` is working, one may for instance check the number of CPUs of each host:

```
dsh -g all -r ssh -Mc "cat /proc/cpuinfo | grep ^processor | wc -l"
```

where `all` is the name of the file where you have the hosts list (`$HOME/.dsh/group/all`).

Installation You do not have to compile anything but you must add the path `utils/ensemble_generation/` to your environment variable `PYTHONPATH`. Then, you can load the module with:

```
from ensemble_generation import *
```

Note: This tools library is written in Python and uses the library AtmoPy. The shell Bash is also required.

2.9.2 Configuration Files

Ensemble Parameters The file `parameter.cfg` is used to design the ensemble. It contains four sections: `general`, `physic`, `numeric` and `input_data`. In the section `general`, you have some general information about the simulation like in the configuration file `general.cfg`:

```
date_min: 2001-04-22 date_max: 2001-04-30

x_min = -10. Delta_x = 2. Nx = 16
y_min = 36. Delta_y = 2. Ny = 11
```

Moreover, you have to specify the number of models in your ensemble and two directories where the preprocessing results (`data_dir`) and the models results (`model_dir`) will be stored:

```
Nmodel = 25

data_dir: /nfs/polyphemus/user/work/ensemble/2001/data
model_dir: /nfs/polyphemus/user/work/2001/ensemble
```

For the sections `physics` and `numerics`, you have to specify the name of the physical parameterizations and the numerical approximations. In the both sections, you have to specify the multiple choices for each parameter. Moreover, it is possible to give the occurrence frequency for each choice. You may add the keyword `preprocessing` to indicate that this parameter takes part to preprocessing.

```
[physics]

luc:                  usgs (50),  glcf (50)  preprocessing
chemistry:            racm (60),  radm (40)
deposition_velocity: zhang, wesely           preprocessing
kz:                   tm - louis             preprocessing


[numerics]

time_step:             600. (90), 1800. (10)
vertical_resolution:  5 (70) - 9 (30)        preprocessing
with_air_density:     yes, no
```

You must specify the occurrence frequency between brackets just after a parameter value. You can separate the values by a coma or a dash. You do not have to specify the occurrence frequency every time. In this case, the random choice will be uniform. Nevertheless you can not specify the occurrence frequency just for one value. For example:

```
deposition_velocity:  zhang (75), wesely      preprocessing
```

is wrong.

For the perturbations of input data, you have to specify the name of the field which will perturbed, the uncertainty encompassing 95% of all possible values and the type of distribution. For a normal distribution, the random choice will be made between the raw field, the raw field minus the standard deviation and the raw field plus the standard deviation.

```
[input_data]

# Name          # Uncertainty      # Law
wind_velocity:  1.3                log-normal
wind_angle:     40.                normal
temperature:    3.                 normal
emissions_NOx:  1.5                log-normal
dep:            2.                 log-normal
```

Each randomly built model has its own identity. The model identity is a list of integers. Each integer represents a physical parameterization, a numerical approximation or a perturbed input field. The value of each integer represents a parameter value. Assume you have this configuration file:

```
[physic]

luc:       usgs,  glcf  preprocessing
chemistry: racm,  radm

[numeric]
```

```
time_step: 600.,  1800.
vertical_resolution: 5,  7,  9  preprocessing
first_layer_height: 40.,  50.   preprocessing

[input_data]

temperature:    3.    normal
wind_angle:     40.   normal
wind_velocity:  1.5   log-normal
emissions_NOx:  1.5   log-normal
```

You have nine different parameters. The model identity `010210221` represents:

0	luc	usgs
1	chemistry	radm
0	time step	600 s
2	vertical resolution	7
1	first layer height	50 m
0	temperature	raw field
2	wind angle	raw field + 20 degrees
2	wind velocity	raw field * 1.24
1	NO_x emissions	raw field * 0.82

Ensemble Programs The configuration file `program.cfg` contains the names of the programs, their dependencies and an integer which gives the group index. A list of programs with a group index equal to 0 will be launched before an other list of program with a group index equal to 1. The dependencies are the names of physical parameterizations or numerical approximations which are going to change the preprocessing results. For example, the deposition velocities from the program `dep` depend on the LUC, the vertical resolution, the first layer height and a physical parameterization (zhang or wesely). The results will depend on the values of each parameter. Each program belongs to a specific section. The section `general` must contain the path to the directory where all generic configuration files are. In the other sections, before each programs list, you have to specify the directory that contains the generic configuration files.

```
[general]

home: /home/garaud
generic_cfg: <home>/src/ensemble_generation/example/configuration


[ground]

config_directory: <generic_cfg>/preprocessing/ground
luc:        luc    0
roughness:  luc    10


[meteo]
```

```
config_directory: <generic_cfg>/preprocessing/meteo
meteo:        vertical_resolution, first_layer_height    20
attenuation:  vertical_resolution, first_layer_height, cloud_attenuation   30
Kz:           luc, vertical_resolution, first_layer_height, min_kz         40
Kz_TM:        luc, vertical_resolution, first_layer_height, min_kz, apply_vert   50


[dep]

config_directory: <generic_cfg>/preprocessing/dep
dep: luc, vertical_resolution, first_layer_height, deposition_velocity     40
```

Each dependency has to match a physical parameterization or a numerical approximation described in the configuration file `parameter.cfg`.

In the generic configuration files, you have a lot of variables between % which will be replaced by certain values according to the model identity. For example, the generic configuration file `meteo.cfg` contains:

```
[paths]
LUC_file: <Directory_ground_data>/%luc_dir%/LUC-<LUC_origin>.bin
Directory_meteo: <Directory_computed_fields>/meteo/%meteo_dir%/
Directory_attenuation: <Directory_computed_fields>/meteo/%attenuation_dir%/

[Kz]
Min = %min_kz%
Apply_vert: %apply_vert%
```

The preprocessing results will be stored in different directories named after the values of the parameters. For example, the program `Kz` may output results in:

```
/home/garaud/results/data/meteo/Kz/usgs/Nz-5/flh-40/min-0.2/
```

2.9.3 Quick Start

Ensemble Generation You can quickly build an ensemble of models with these three Python lines:

```
from ensemble_generation import *

p = EnsembleParameter('parameter.cfg')
p.GenerateIdEnsemble()
```

You can save your ensemble identities in a file or load them from a file with these two methods:

```
p.WriteIdEnsembleFile('id_ensemble.dat')
p.ReadIdEnsembleFile('id_ensemble.dat', copy = True)
```

Programs Launching You can launch several Polyphemus programs with the class `EnsembleProgram`. First, you can implement an object based on `ConfigReplacement` in the file `modules/ConfigReplacement.py`. This class contains four different methods which return dictionaries:

- `GetConfigVariable`
  - `keys`: The variables between `%` in the generic configuration files.
  - `values`: The names of the parameters values and the directories where the preprocessing results will be stored. These values depend on the model identity.
- `GetDefaultDict`
  - `keys`: The names of the parameters in the configuration file `parameter.cfg`
  - `values`: The default values of parameters which would not appear in the configuration file.
- `GeBinaryFile`
  - `keys`: The names of the programs.
  - `values` The names of the binary files generated by the programs.
- `GetPerturbedFieldList`
  - `keys`: The variables between `%` in the generic configuration file `perturbation.cfg`.
  - `values`: The name of the perturbed fields, the uncertainties and the types of distribution (normal or log-normal).

For example:

```
In[1]: d = ConfiReplacement.GetConfigVariable(model_index = 0, ensemble_program)

In[2]: d
Out[2]:
{'%Date%': '20010422',
'%Nx%': '16',
'%Ny%': '11',
'%luc%': 'usgs',
'%dep%': 'wesely',
'%luc_dir%': 'luc/usgs',
'%dep_dir%': 'dep/usgs/Ns-5/wesely'
}
```

Your class `ConfigReplacement` must contain these four methods and an empty constructor. Then, you can launch your Polyphemus programs:

```
from ensemble_generation import *

# Reads the configuration files.
epr = EnsembleProgram('parameter.cfg', 'program.cfg',
                      only_preprocessing = True)
```

```
# Sets the Polyphemus directory.
epr.SetPolyphemusDirectory('/home/garaud/src/polyphemus')

# Generates a new ensemble.
epr.parameter.GenerateIdEnsemble()

# Sets the ConfigReplacement object.
config_replacement = modules.ConfigPolair3D()
epr.SetConfigReplacement(config_replacement)

# Gets an instance 'Polyphemus' and creates all directories where you
# will have the results.
ens = epr.GetEnsemble(group = 'polyphemus')

# Gets the available hosts from your hosts list (in the file
# '$HOME/.dsh/group/polyphemus').
load_averages = ens.net.GetAvailableHosts()

# You can launch all programs.
ens.RunNetwork()
```

Chapter 3

Preprocessing

This chapter introduces all preprocessing programs. It details the input files (data files and configuration files) of every program, and it describes their output files. In Section 3.2, configurations and features shared by almost all programs are explained.

3.1 Remark

In the descriptions of preprocessing programs, there are references to functions like `ComputePressure`, `ComputeAttenuation_LWC`, etc. These functions are part of AtmoData and are described in AtmoData scientific documentation [Njomgang et al., 2005].

3.2 Introduction

3.2.1 Running Preprocessing Programs

Most preprocessing programs:

- accept one or two configuration files as arguments;
- *append* their results at the end of binary files (if they already exist) or create them. Note that they cannot create the directory so you have to make sure it exists before launching a preprocessing program.

For instance, program `meteo` processes meteorological data over one day. To generate data from day 2001-05-19 to day 2001-05-21, one should launch:

```
./meteo ../general.cfg meteo.cfg 2001-05-19
./meteo ../general.cfg meteo.cfg 2001-05-20
./meteo ../general.cfg meteo.cfg 2001-05-21
```

Another option is to use the script `call_dates` (see Section 2.8.4). In that case, to generate data from day 2005-05-19 to day 2005-05-21, one should launch:

```
../../utils/call_dates ./meteo ../general.cfg meteo.cfg 20050519 20050521
```

or

```
../../utils/call_dates ./meteo ../general.cfg meteo.cfg 20050519 3
```

Remember that *the results are appended at the end of the output files if they already exist.* If you decide to recompute your fields from the first day, you have to *first remove* old output binary files.

In order to know what are the arguments of a program, you may launch it without arguments. For instance:

```
emissions> ./emissions

Usage:
  ./emissions [main configuration file] [secondary configuration file] [date]
  ./emissions [main configuration file] [date]
  ./emissions [date]

Arguments:
  [main configuration file] (optional): main configuration file. Default: emissions.cfg
  [secondary configuration file] (optional): secondary configuration file.
  [date]: date in any valid format.
```

3.2.2 Configuration

Almost all programs require the description of the domain over which computations should be performed. Since this configuration is shared by many programs, it is put in a common configuration file called `general.cfg`. An example of such a file is `Polyphemus/preprocessing/general.cfg`, whose content is quoted below:

```
[general]

Home: /u/cergrene/0/bordas
Directory_computed_fields: <Home>/B/data
Directory_ground_data: <Directory_computed_fields>/ground
Programs: <Home>/codes/Polyphemus-HEAD


[domain]

Date: 2001-01-02
Delta_t = 3.0
x_min = -10.0   Delta_x = 0.5   Nx = 65
y_min = 40.5    Delta_y = 0.5   Ny = 33
Nz = 5
Vertical_levels: <Programs>/levels.dat
```

Entries in section `[general]` are markups provided for convenience. See Section 2.5.2 for further explanations.

The section `[domain]` contains the domain (in space and time) description.

`[domain]`	
`Date`	The date at which the simulation (of the chemistry-transport model) is starting. It is also the date at which meteorological data (processed by Polyphemus – output from programs `meteo`, `MM5-meteo` or `WRF-meteo`) starts. As a consequence, any program that needs to read this meteorological data refers to this date. The date must be in a format described in Section D.7.
`Delta_t`	Time step in hour of *output meteorological data* processed by Polyphemus.
`x_min`	Abscissa of the center of the lower-left cell. It is usually in longitude (degrees).
`Delta_x`	Step length along x, usually in degrees (longitude).
`Nx`	Number of cells along x (integer).
`y_min`	Ordinate of the center of the lower-left cell. It is usually in latitude (degrees).
`Delta_y`	Step length along y, usually in degrees (latitude).
`Ny`	Number of cells along y (integer).
`Nz`	Number of vertical levels (integer).
`Vertical_levels`	Path to the file that defines vertical levels interfaces in m.

3.2.3 Dates

Many preprocessing programs require a starting and end date. In that case, the starting date must always be provided in command line, but there are three possibilities for the end date:

1. provide an end date in any valid format (see Section D.7), e.g.

   ```
   ./dep ../general.cfg dep.cfg 2004-08-09_12-00-00 2004-08-11_06-00-00
   ```

2. provide a duration in format `NdMh` or `Nd-Mh` for N *days* and M *hours*. Alternatively, you can put a duration of `Mh` for M *hours* or `Nd` for N *days*. Valid duration can be, for instance, `3d5h`, `2d-8h`, `5d` or `14h`, e.g.

   ```
   ./dep ../general.cfg dep.cfg 2004-08-09_12-00-00 1d-12h
   ```

3. provide no end date nor duration. In that case a duration of one day is used, e.g.

   ```
   ./dep ../general.cfg dep.cfg 2004-08-09_12-00-00
   ```

Please note that in all three cases the end date is excluded. The advantage of it is that, if for whatever reason you want to use the preprocessing program several times in a row, you can use the end date as the starting date for the next time.

The number of iterations is computed from the interval between the starting and end dates and from the time step you provided in the general configuration file.

The first date given in command line is the date at which the preprocessing starts *this time*. It might differ from the date given in `general.cfg` if the preprocessing program is launched several times in a row, for example because the meteorological data file does not cover the whole time span of the simulation.

3.2.4 Data Files

Polyphemus reads ECMWF Grib files, MM5 files, NetCDF files (for WRF files or Mozart 2), text files and binary files. All files generated by Polyphemus are text files or binary files.

Unless specified otherwise, all binary files store single-precision floating-point numbers. They do not contain any header. Each binary file only stores the values of a single field. Four-dimensional fields are stored this way:

```
Loop on time t
   Loop on z
      Loop on y
         Loop on x
```

Let this storage be symbolized by $\{t, z, y, x\}$. Dimensions t, z, y and x always appear in this order. For instance, three-dimensional fields may be stored in formats $\{z, y, x\}$ or $\{t, y, x\}$, $\{t, z, x\}$ or $\{t, z, y\}$.

3.3 Ground Data

Computing ground data is the first step of a preprocessing as they are necessary to process meteorological fields. All programs related to ground-data generation are available in `Polyphemus/preprocessing/ground`.

The first step should be program `luc-usgs` or `luc-glcf` depending on what raw data you have. Land use data may come from the US Geological Survey (USGS) or from the Global Land Cover Facility (GLCF).

3.3.1 Land Use Cover – GLCF: `luc-glcf`

In order to prepare land use cover from GLCF, one should use program `luc-glcf`. It is recommended to download the global land use cover file at 1 km resolution provided in latitude–longitude coordinates and with extension `.bsq`.

At the time this documentation is written, the file is available at:
`ftp://ftp.glcf.umiacs.umd.edu/glcf/Global_Land_Cover/Global/gl-latlong-1km-landcover/gl-latlong-1km-landcover.bsq.gz` (single line, no white space – you may use `wget` to download it, or copy and paste the URL in your favorite browser).

You need to uncompress this file (e.g., `gunzip gl-latlong-1km-landcover.bsq.gz`).

In case the file has been moved, try to find it from `http://glcfapp.glcf.umd.edu:8080/esdi/index.jsp`. Select "Product Search", then "AVHRR, Global Land Cover Product" and finally use:

- Region: Global
- Projection: Lat/Long
- Resolution: 1 km

Click on "Download" and select the file `gl-latlong-1km-landcover.bsq.gz`.

Finally you have to fill the configuration file `luc-glcf.cfg`. Note that the default values in section `[GLCF]` are for file `gl-latlong-1km-landcover.bsq`: there is no need to change them if you downloaded this recommended file.

	[paths]
Database_luc-glcf	Directory where the raw data from GLCF can be found (directory where gl-latlong-1km-landcover.bsq lies).
LUC_in	Name of the file containing raw data (i.e. gl-latlong-1km-landcover.bsq or its new name if you renamed it).
Directory_luc-glcf	Output directory.
LUC_out	Output filename. The default filename LUC-glcf.bin is recommended for clarity.
	[GLCF]
Step	Space step in degrees in GLCF input file.
x_min	Minimum longitude in the input file (degrees).
y_min	Minimum latitude in the input file (degrees).
Nx	Number of cells along longitude in the input file.
Ny	Number of cells along latitude in the input file.
Nc	Number of land use categories.

The output land-cover file is in format $\{c, y, x\}$ where c stands for (land use) category.

Table 3.3 presents land-use categories as they are computed with luc-glcf.

Table 3.3: Land-use categories in GLCF description.

Value	Label
0	Water.
1	Evergreen Needleleaf Forest.
2	Evergreen Broadleaf Forest.
3	Deciduous Needleleaf Forest.
4	Deciduous Broadleaf Forest.
5	Mixed Forest.
6	Woodland.
7	Wooded Grassland.
8	Closed Shrubland.
9	Open Shrubland.
10	Grassland.
11	Cropland.
12	Bare Ground.
13	Urban and Built.

Program luc-glcf does not require any date as an input in command line. To launch luc-glcf, just type:

```
./luc-glcf ../general.cfg luc-glcf.cfg
```

3.3.2 Land Use Cover – USGS: luc-usgs

For a simulation over Europe, program luc-usgs requires two files found at http://edcsns17.cr.usgs.gov/glcc/:

- USGS Land Use/Land Cover Scheme for Eurasia in Lambert Azimuthal Equal Area Projection (optimized for Europe) available at http://edcftp.cr.usgs.gov/pub/data/glcc/ea/lamberte/eausgs2_0le.img.gz in compressed format.
- USGS Land Use/Land Cover Scheme for Africa in Lambert Azimuthal Equal Area Projection available at http://edcftp.cr.usgs.gov/pub/data/glcc/af/lambert/afusgs2_0l.img.gz in compressed format.

The configuration file `luc-usgs.cfg` requires:

	`[paths]`
`Database_luc-usgs`	Directory where the raw data from USGS can be found.
`LUC_in_ea`	Input file containing raw data for Eurasia (eausgs2_0le.img).
`LUC_in_af`	Input file containing raw data for Africa (afusgs2_0l.img).
`Directory_luc-usgs`	Output directory.
`LUC_out`	Output file name. The default filename `LUC-usgs.bin` is recommended for clarity.
	`[USGS]`
`Step`	Space step in meters.
`lon_origin_ea`	Longitude of the center of lower-right cell for Eurasia.
`lat_origin_ea`	Latitude of the center of the lower-right cell for Eurasia.
`lon_origin_af`	Longitude of the center of the lower-right cell for Africa.
`lat_origin_af`	Latitude of the center of the lower-right cell for Africa.
`lon_upper_left_ea`	Longitude of the center of the upper-left cell for Eurasia.
`lat_upper_left_ea`	Latitude of the center of the upper-left cell for Eurasia.
`lon_upper_left_af`	Longitude of the center of the upper-left cell for Africa.
`lat_upper_left_af`	Latitude of the center of the upper-left cell for Africa.
`Nx_ea`	Number of cells along longitude in the input file for Eurasia.
`Nx_af`	Number of cells along longitude in the input file for Africa.
`Ny_ea`	Number of cells along latitude in the input file for Eurasia.
`Ny_af`	Number of cells along latitude in the input file for Africa.
`Nc`	Number of land categories.
`Sea_index`	Index of the sea in land categories (normally 15).

The output land-cover file is in format $\{c, y, x\}$ where c stands for (land use) category.

Table 3.5 presents land-use categories as they are computed with `luc-usgs`, that is to say with indices starting at 0.

Table 3.5: Land-use categories in USGS description.

Value	Label
0	Urban and Built-Up Land.
1	Dryland Cropland and Pasture.
2	Irrigated Cropland and Pasture.
3	Mixed Dryland/Irrigated Cropland and Pasture.
4	Cropland/Grassland Mosaic.
5	Cropland/Woodland Mosaic.
6	Grassland.

7	Shrubland.
8	Mixed Shrubland/Grassland.
9	Savanna.
10	Deciduous Broadleaf Forest.
11	Deciduous Needleleaf Forest.
12	Evergreen Broadleaf Forest.
13	Evergreen Needleleaf Forest.
14	Mixed Forest.
15	Water Bodies.
16	Herbaceous Wetland.
17	Wooded Wetland.
18	Barren or Sparsely Vegetated.
19	Herbaceous Tundra.
20	Wooded Tundra.
21	Mixed Tundra.
22	Bare Ground Tundra.
23	Snow or Ice.

Program `luc-usgs` does not require any date as an input in command line. To launch `luc-usgs`, just type:

```
./luc-usgs ../general.cfg luc-usgs.cfg
```

3.3.3 Conversions: `luc-convert`

The output of `luc-glcf` or `luc-usgs` are land use cover described with GLCF or USGS categories. It is often useful to convert these descriptions to another set of land use categories. This means, for example, summing up the contributions of sparsely vegetated and bare ground tundra (USGS categories #19 and #22) to estimate the proportion of barren land in Wesely description (category #8). An input category may also be split into several output categories. In practice, one may want to convert in Wesely or Zhang land use cover using `luc-convert`. In particular it is necessary to convert land data from USGS or GLCF to Zhang categories before computing deposition velocities with program `dep` (see Section 3.5.1).

In addition to the domain definition (Section 3.2.2), below is the information required in the configuration file (or configuration files) for `luc-convert`:

	`[paths]`
`Database_luc-convert`	Directory where the input file (input land use categories) is located.
`File_in`	Input file name (in `Database_luc-convert`).
`Directory_luc-convert`	Directory where the output file (output land use categories) should be stored.
`File_out`	Output file name (in `Directory_luc-convert`).
	`[dimensions]`
`Nc_in`	Number of land categories in the input format.
`Nc_out`	Number of land categories in the output format.
	`[coefficients]`
Correspondence matrix between input land categories and output land categories. Each line corresponds to an input category. Each line contains: the index of the category	

(or any number: this first column is not read) and the distribution of the input category in all output categories (columns). The distribution is a set of numbers in $[0, 1]$ whose sum should be 1.

Several configuration files are provided to convert GLCF or USGS categories to Wesely or Zhang categories: `glcf_to_wesely.cfg`, `glcf_to_zhang.cfg`, `usgs_to_wesely.cfg` and `usgs_to_zhang.cfg`.

The output land-cover file is in format $\{c, y, x\}$ where c stands for (land use) category.

The conversion can be launched with:

```
./luc-convert ../general.cfg usgs_to_wesely.cfg
```

3.3.4 Roughness: `roughness`

After land use cover has been computed, roughness data can be estimated, using program `roughness`.

	`[domain]`
`Nx`	Number of grid points along longitude.
`Ny`	Number of grid points along latitude.
	`[paths]`
`LUC_file`	File where the land use cover data are stored (e.g., computed using `luc-glcf` or `luc-usgs`).
`Directory_roughness`	Directory where the output file will be stored.
`Roughness_out`	Output file name.
	`[data]`
`Roughness_data_file`	Path to the file giving the roughness of land categories. This file should be a text file with three columns: the land category index (starting at 0), the roughness height (in m) and the category name. Two examples are provided: `roughness-glcf.dat` and `roughness-usgs.dat`

A field named `LUC_origin` is also defined. It is used for markup substitution in the various filenames. The value of this field must be `glcf` or `usgs`.

The program may be launched with:

```
./roughness ../general.cfg roughness.cfg
```

Section `[domain]` is in `general.cfg` and the other sections are read in `roughness.cfg`.

3.3.5 LUC for emissions: `extract-glcf`

This program is only necessary in order to generate anthropogenic emissions from EMEP inventories (see Section 3.6.2). The output is the land category (read from GLCF global land-use classification, `gl-latlong-1km-landcover.bsq`) over a subdomain. Make sure that the output domain (described in section `[subdomain]` of the configuration file) entirely contains your simulation domain.

The configuration file should contain (see example `extract-glcf.cfg`):

	[paths]
File_GLCF	GLCF input file (global). It is the same as the one used for luc-glcf.
File_out	Output file. That is, the file given in section [LUC] of emissions.cfg.
	[GLCF]
x_min	Minimum longitude of the GLCF domain (whole world).
Nx	Number of steps along the longitude.
y_max	Maximum latitude of the GLCF domain (whole world).
Ny	Number of steps along the latitude.
Step	Space step (in degrees) of the input GLCF file. The output file will have the same resolution.
	[subdomain]
x_min	Minimum longitude of the subdomain.
Nx	Number of steps along the longitude.
y_min	Minimum latitude of the subdomain.
Ny	Number of steps along the latitude.

The program may be launched with

```
./extract-glcf ../general.cfg extract-glcf.cfg
```

Note that the file `../general.cfg` is not compulsory providing no markup from it is used in `extract-glcf.cfg`.

The output is a binary file of integers between 0 and 13.

3.4 Meteorological Fields

3.4.1 Program `meteo`

Program `Polyphemus/preprocessing/meteo/meteo` reads ECMWF Grib files and generates meteorological fields required by chemistry-transport models. Most fields are interpolated from ECMWF grid to a regular grid (latitude/longitude in the horizontal, altitudes in meters in the vertical). It is assumed that ECMWF input data are stored in daily Grib files. That is why this program (as well as `attenuation` and `Kz`) only processes data daily and requires only one date as an input in command line. To extract Grib files, Polyphemus uses the WGRIB package whose a compatible version should now be included in our distribution package as explained in Section 1.3.4.

Note that `meteo` needs as input data the land use cover which can be built using programs in `preprocessing/ground/`.

The reference configuration files for `meteo` is `Polyphemus/preprocessing/meteo/meteo.cfg` together with `Polyphemus/preprocessing/general.cfg`.

In addition to the domain definition, below are options of `meteo`:

	`[paths]`
`Database_meteo`	Directory in which ECMWF input files may be found. It is assumed that ECMWF file names are in format `ECMWF-YYYYMMDD` where `YYYY` is the year, `MM` the month and `DD` the day. If program `meteo` is launched for a day D, ECMWF data files for days D-1 and D must be available. Data for day D-1 are needed to process cumulated data (e.g., solar radiation).
`Roughness_in`	Path to the binary file that describes roughness heights (in meters) in ECMWF grid cells. Its format is $\{y, x\}$. It is needed only if option `Richardson_with_roughness` is activated.
`Directory_meteo`	Directory where output meteorological files are stored.
	`[ECMWF]`
`Date`	Date at which the meteorological file begins. It is referred as `&D` which is the date given in command line because it is supposed that ECMWF Grib files store data daily.
`t_min`	First hour stored in the Grib file.
`Delta_t`	Time step (in hour) of data stored in every ECMWF file.
`Nt`	Number of time steps stored in every ECMWF file.
`x_min`	Longitude in degrees of the center of the lower-left cell in ECMWF grid.
`Delta_x`	Step length (in degrees) along longitude of ECMWF grid.
`Nx`	Number of cells along longitude (integer) in ECMWF grid.
`y_min`	Latitude in degrees of the center of the lower-left cell in ECMWF grid.
`Delta_y`	Step length (in degrees) along latitude of ECMWF grid.
`Ny`	Number of cells along latitude (integer) in ECMWF grid.
`Nz`	Number of vertical layers (integer) in ECMWF grid.
	`[meteo]`
`Richardson_with_roughness`	Should the surface Richardson number be computed taking into account roughness height?
	`[accumulated_data]`
`Accumulated_time`	For data storing values cumulated in time (e.g., solar radiation), length number of time steps over which data is cumulated.
`Accumulated_index`	Start index of the first complete cycle of cumulated data. Data is then cumulated from `t_min` plus `Accumulated_index` times `Delta_t`.

To launch the program, just type :

```
./meteo ../general.cfg meteo.cfg 2001-04-22
```

The program basically reads data in the ECMWF Grib file and interpolates it in time and space to Polyphemus grid. ECMWF data is described in `meteo.cfg` and Polyphemus grid is described in `general.cfg`. For the sake of simplicity, it is recommended to work with ECMWF files containing data for one day. Program `meteo` should be called for each day (preferably from

0h to 24h), that is, for each available ECMWF file (except the first one – see below). If ECMWF files are not provided on a daily basis, it is recommended to contact the Polyphemus team at polyphemus-help@lists.gforge.inria.fr.

In order to process the ECMWF file for a given day, the ECMWF file for the previous day must be available. Indeed, ECMWF files contain data that is accumulated over several time steps (like rain), and values from previous steps (including from the previous day) must be subtracted to get the actual value of the field.

Here is the list of input data needed in ECMWF files (with their Grib code): surface temperature (167), skin temperature (235), surface pressure (152), temperature (130), specific humidity (133), liquid water content (246), medium cloudiness (187), high cloudiness (188), meridional wind (132), zonal wind (131), zonal friction velocity (180), meridional friction velocity (181), solar radiation (169), boundary layer height (159), soil water content (39), sensible heat (146), evaporation (182).

The list of output variables is: pressure, surface pressure, temperature, surface temperature, skin temperature, Richardson number, surface Richardson number, specific humidity, liquid water content, solar radiation, photosynthetically active radiations (direct beam, diffuse and total), zonal wind, meridional wind, wind module, wind friction module, boundary layer height, soil water content, evaporation, sensible heat and first-level wind module.

Inside `meteo`, ECMWF variables are read and decumulated (in time) if necessary. Pressures at ECMWF levels are computed with `ComputePressure` and altitudes are computed with `ComputeInterfHeight` and `ComputeMiddleHeight`. Richardson number is then estimated with `ComputeRichardson`. All input fields are then interpolated on the vertical. Finally photosynthetically active radiation are estimated, based on solar radiation and zenith angle (`ZenithAngle`).

To get the complete set of input meteorological data for a transport model, one should then launch `attenuation`, `Kz` and maybe `Kz_TM`.

3.4.2 Program `attenuation`

Program `Polyphemus/preprocessing/attenuation` should be launched after program `meteo`. It computes cloud attenuation for photolysis rates. It also computes cloud related data, such as cloud height. Even for passive simulations this program should be launched.

The reference configuration files for `attenuation` is `preprocessing/meteo/meteo.cfg` together with `preprocessing/general.cfg`. In addition to the domain definition and to the entries of `meteo.cfg` introduced in Section 3.4.1, below are options for `attenuation`:

	`[paths]`
`Directory_attenuation`	Directory where the output of program `attenuation` is stored.
	`[attenuation]`
`Type`	Parameterization to be used to compute cloud attenuation. Put `1` to use `RADM` parameterization or put `2` to use `ESQUIF` parameterization.
`Critical_relative_humidity`	Parameterization to compute critical relative humidity. Put `1` to compute CRH with respect to σ (recommended) or put `2` to have a constant CRH in two distinct layers.

	[clouds]
Min_height	Minimum cloud basis height in m.

Just like in program meteo, ECMWF data is read and interpolated. Then the relative humidity and the critical relative humidity are computed respectively with ComputeRelativeHumidity and ComputeCriticalRelativeHumidity. The cloud fraction is computed with ComputeCloudFraction. For it the cloudiness and cloud height are diagnosed using ComputeCloudiness and ComputeCloudHeight. Finally attenuation coefficients are computed with ComputeAttenuation_LWC (RADM parameterization) or ComputeAttenuation_ESQUIF (ESQUIF parameterization).

Output files are:

- the rain intensity (Rain.bin) in $\mathrm{mm\,h^{-1}}$,
- the convective rain intensity (ConvectiveRain.bin) in $\mathrm{mm\,h^{-1}}$,
- the 3D cloud attenuation coefficient (in $[0, 2]$),
- the cloud height in m.

To launch the program, just type :

```
./attenuation ../general.cfg meteo.cfg 2001-04-22
```

3.4.3 Program Kz

Program Kz computes the vertical diffusion coefficients (needed in almost all applications) using Louis parameterization [Louis, 1979]. It should be launched after attenuation.

The reference configuration files for Kz is Polyphemus/preprocessing/meteo/meteo.cfg together with Polyphemus/preprocessing/general.cfg. In addition to the domain definition and to the entries of meteo.cfg introduced in Section 3.4.1 and 3.4.2, below are options for Kz:

	[paths]
File_Kz	Name of the file where the vertical diffusion coefficients (output) are stored.
	[Kz]
Min	Lower threshold for vertical diffusion in $\mathrm{m^2\,s^{-1}}$.
Min_urban	Lower threshold for vertical diffusion over urban areas, in $\mathrm{m^2\,s^{-1}}$.
Max	Higher threshold for vertical diffusion in $\mathrm{m^2\,s^{-1}}$.
Apply_vert	If set to no, the lower threshold is applied only at the top of the first layer, otherwise it is applied to all levels.

This programs mainly computes the vertical diffusion coefficients with a call to ComputeLouisKz. Simple corrections are also performed to take into account convective conditions.

The output is a 3D time-dependent field (format $\{t, z, y, x\}$) of vertical diffusion coefficients (in $\mathrm{m^2\,s^{-1}}$). Along the vertical, the coefficients are defined on the interfaces. So the size of the field (for each day) is $4 \times Nt \times (Nz + 1) \times Ny \times Nx$. It is stored in the path given by entry File_Kz.

To launch the program, just type :

```
./Kz ../general.cfg meteo.cfg 2001-04-22
```

3.4.4 Program Kz_TM

Program Kz_TM "overwrites", in the boundary layer height, the vertical diffusion coefficients computed with Louis parameterization, with coefficients computed according to Troen & Mahrt parameterization [Troen and Mahrt, 1986]. It should be launched either after Kz, after MM5-meteo or after WRF-meteo.

The reference configuration files for Kz_TM is Polyphemus/preprocessing/meteo/meteo.cfg or Polyphemus/preprocessing/meteo/MM5-meteo.cfg or Polyphemus/preprocessing/meteo/WRF-meteo.cfg together with Polyphemus/preprocessing/general.cfg.

In addition to the domain definition in general.cfg, below are the entries for Kz_TM:

	[path]
LUC_file	Path to the binary file that describes land use cover over the *output* grid (described in section [domain]). This file must be in format $\{l, y, x\}$ (l is the land category) and it must contain proportions (in $[0, 1]$) of each land category in every grid cell.
Sea_index	Index of sea in land categories. It is 0 for GLCF description and 15 for USGS description.
Urban_index	Index of cities in land categories. It is 13 for GLCF description and 0 for USGS description.
Roughness_file	Path to the binary file that describes roughness heights (in meters) in output grid cells. Its format is $\{y, x\}$. It is needed only if option Flux_diagnosed is activated.
Directory_meteo	Directory where output meteorological files are stored.
File_Kz	Name of the file where the vertical diffusion coefficients as computed with the Louis parameterization are stored.
Directory_Kz_TM	Name of the directory where the vertical diffusion coefficients (output) are stored (the filename being Kz_TM.bin).
	[Kz]
Min	Lower threshold for vertical diffusion in $m^2\,s^{-1}$.
Min_urban	Lower threshold for vertical diffusion over urban areas, in $m^2\,s^{-1}$.
Max	Higher threshold for vertical diffusion in $m^2\,s^{-1}$.
Apply_vert	If set to no, the lower threshold is applied only at the top of the first layer, otherwise it is applied to all levels.
p	Coefficient used in Troen and Mahrt parameterization (see Troen and Mahrt [1986]).
C	Coefficient used in Troen and Mahrt parameterization (see Troen and Mahrt [1986]).
SBL	Ratio between the surface layer and the atmospheric boundary layer (0.1 in Troen and Mahrt [1986]).
Ric	Critical Richardson number used to estimate the atmospheric boundary layer height (in case BL_diag is set to 2).
Fluxes_diagnosed	Should the friction module, the evaporation and the sensible heat be diagnosed? If not, they are read in input data (which is recommended).

BL_diag	What kind of diagnosis is used to estimate the boundary layer height? Put 1 to use Troen and Mahrt diagnosis [Troen and Mahrt, 1986]; put 2 to rely on a critical Richardson number; and put 3 to use ECMWF (or MM5 or WRF) boundary layer height (so, there is no diagnosis – this option is more robust and it is recommended).
Perturbed_BL	Multiplication factor for the boundary layer height in the Troen & Mahrt parameterization.
TM_stable	The vertical diffusion as computed by Troen and Mahrt parameterization is applied only within the boundary layer. It is possible to further restrict its application: if TM_stable is set to no, the parameterization is not applied in stable conditions. In this case, the Troen and Mahrt parameterization is only applied in unstable boundary layer.

Several meteorological fields are computed with ComputePotentialTemperature, ComputeSaturationHumidity and ComputeSurfaceHumidity_diag. If fluxes are not diagnosed, the Monin-Obukhov length is computed with ComputeLMO. Then the boundary layer height may be diagnosed with ComputePBLH_TM (Troen & Mahrt parameterization) or ComputePBLH_Richardson (critical Richardson number). Finally the vertical diffusion coefficients are computed with ComputeTM_Kz.

The main output is a 3D time-dependent field (format $\{t, z, y, x\}$) of vertical diffusion coefficients (in $m^2\,s^{-1}$). Along the vertical, the coefficients are defined on the interfaces. So the size of the field (for each day) is $Nt \times (Nz + 1) \times Ny \times Nx$. It is stored in Kz_TM.bin in the directory given by entry Directory_Kz_TM. The surface relative humidity is saved in SurfaceRelativeHumidity.bin. Depending on the options, additional fields may be saved, such as the Monin-Obukhov length in file LMO.bin.

3.4.5 Program MM5-meteo

Program Polyphemus/preprocessing/meteo/MM5-meteo processes MM5 data and generates meteorological fields required by chemistry-transport models. Most fields are interpolated from MM5 grid to a regular grid (latitude/longitude in the horizontal, altitudes in meters in the vertical).

Note that MM5-meteo needs as input data the land use cover which can be built using programs in preprocessing/ground.

Note for ECMWF users: program MM5-meteo is equivalent to what is performed by meteo, attenuation and Kz successively. Similarly to ECMWF files, Kz_TM can be used afterwards.

Program MM5-meteo can be launched as follows:

```
./MM5-meteo ../general.cfg MM5-meteo.cfg 2004-08-09_09-00-00
./MM5-meteo ../general.cfg MM5-meteo.cfg 2004-08-09_09-00-00 2004-08-10_09-00-00
./MM5-meteo ../general.cfg MM5-meteo.cfg 2004-08-09_09-00-00 1d
```

The configuration file MM5-meteo.cfg contains several options:

	[paths]
Database_MM5_meteo	Directory in which MM5 input files may be found. If &D appears in the file name, it is replaced by YYYY-MM-DD where YYYY is the year, MM the month and DD the day of the date given in command line.
Directory_meteo	Directory where output meteorological files are stored.
File_Kz	Name of the file where the vertical diffusion coefficients as computed with the Louis parameterization are stored.
	[MM5]
Delta_t	Time step (in hour) of data stored in every MM5 file.
Nt	Number of time steps stored in every MM5 file.
x_min	Index in MM5 coordinates of the center of the lower-left cell in MM5 grid. This is most likely 0.5.
Delta_x	Index (MM5 coordinates) increase along longitude of MM5 grid. This is most likely 1.
Nx	Number of cells (or dot points) along longitude (integer) in MM5 grid.
y_min	Index in MM5 coordinates of the center of the lower-left cell in MM5 grid. This is most likely 0.5.
Delta_y	Index (MM5 coordinates) increase along latitude of MM5 grid. This is most likely 1.
Ny	Number of cells (or dot points) along latitude (integer) in MM5 grid.
Nz	Number of vertical layers (integer) in MM5 grid.
projection_type	Type of projection. 1 corresponds to Lambert conformal conic, 2 to Mercator and 3 to stereographic.
	[accumulated_rain]
Prev_accumulated_rain	Is the rain accumulated from the previous day?
	[attenuation]
Type	Parameterization to be used to compute cloud attenuation. Put 1 to use RADM parameterization or put 2 to use ESQUIF parameterization.
	[clouds]
Min_height	Minimum cloud basis height in m.
	[Kz]
Min	Lower threshold for vertical diffusion in $m^2\,s^{-1}$.
Min_urban	Lower threshold for vertical diffusion over urban areas, in $m^2\,s^{-1}$.
Max	Higher threshold for vertical diffusion in $m^2\,s^{-1}$.
Apply_vert	If set to no, the lower threshold is applied only at the top of the first layer, otherwise it is applied to all levels.

The program basically reads data in MM5 output file and interpolates it in time and space to Polyphemus grid. MM5 file is described in MM5-meteo.cfg and Polyphemus grid is described

in `general.cfg`. Note that each time a field is loaded by `MM5-meteo`, all time steps are loaded in memory. Note that the fields are released from memory when unused, but you may still need a lot of memory for big MM5 output files.

The program first computes the altitude of MM5 layers, and converts the Polyphemus grid coordinates (latitude/longitude) to MM5 grid coordinates (Lambert, Mercator or stereographic) for interpolations. Interpolations on the horizontal are performed in MM5 grid for efficiency. The pressure is computed based on MM5 fields. The winds are rotated: this gives meridional and zonal winds. The Richardson number is then computed (`ComputeRichardson`). The relative humidity and the critical relative humidity are computed respectively with `ComputeRelativeHumidity` and `ComputeCriticalRelativeHumidity`. The cloud fraction is computed with `ComputeCloudFraction`. For it the cloudiness and cloud height are diagnosed using `ComputeCloudiness` and `ComputeCloudHeight`. Finally attenuation coefficients are computed with `ComputeAttenuation_LWC` (RADM parameterization) or `ComputeAttenuation_ESQUIF` (ESQUIF parameterization). The vertical diffusion coefficients are computed with `ComputeLouisKz` [Louis, 1979]. Finally photosynthetically active radiation are estimated, based on solar radiation and zenith angle (`ZenithAngle`).

Among output files one may find:

- the pressure and the surface pressure in Pa,
- the temperature, the surface temperature and the skin temperature in K,
- the meridional and zonal winds (`MeridionalWind.bin` and `ZonalWind.bin`) in $\mathrm{m\,s^{-1}}$,
- the Richardson number and the surface Richardson number,
- the boundary layer height in m,
- the vertical diffusion coefficients (time-dependent 3D field, defined on layer interfaces on the vertical, `Kz_Louis.bin`) in $\mathrm{m^2\,s^{-1}}$,
- the specific humidity in $\mathrm{kg\,kg^{-1}}$,
- the liquid water content in $\mathrm{kg\,kg^{-1}}$,
- the cloud attenuation coefficients (3D field, `Attenuation.bin`) in $[0, 2]$,
- the solar radiation intensity (`SolarRadiation.bin`) in $\mathrm{W\,m^{-2}}$,
- the rain intensity (`Rain.bin`) in $\mathrm{mm\,h^{-1}}$,
- the convective rain intensity (`ConvectiveRain.bin`) in $\mathrm{mm\,h^{-1}}$,
- the cloud height in m.

3.4.6 Program `MM5-meteo-castor`

Program `MM5-meteo-castor` processes MM5 data and generates meteorological fields required by chemistry-transport model Castor.

	[domain]
Vertical_levels	File containing the parameters alpha and beta used to compute the pressure at various levels and the altitudes. **Note that for most preprocessing programs, this field designates file** preprocessing/levels.dat **but not for this specific application, for which you can use file** preprocessing/meteo/hybrid_coefficients.dat**.**
	[paths]
Database_MM5_meteo	Directory in which MM5 input files may be found. If &D appears in the file name, it is replaced by YYYY-MM-DD where YYYY is the year, MM the month and DD the day.
Roughness_file	Path to the binary file that describes roughness heights (in meters) per month in output grid cells. Note that this file is not the output of program roughness.
Directory_meteo	Directory where output meteorological files are stored.
	[MM5]
Delta_t	Time step (in hour) of data stored in every MM5 file.
Nt	Number of time steps stored in every MM5 file.
x_min	Index in MM5 coordinates of the center of the lower-left cell in MM5 grid. This is most likely 0.5.
Delta_x	Index (MM5 coordinates) increase along longitude of MM5 grid. This is most likely 1.
Nx	Number of cells (or dot points) along longitude (integer) in MM5 grid.
y_min	Index in MM5 coordinates of the center of the lower-left cell in MM5 grid. This is most likely 0.5.
Delta_y	Index (MM5 coordinates) increase along latitude of MM5 grid. This is most likely 1.
Ny	Number of cells (or dot points) along latitude (integer) in MM5 grid.
Nz	Number of vertical layers (integer) in MM5 grid.
projection_type	Type of projection. 1 corresponds to Lambert conformal conic, 2 to Mercator and 3 to stereographic.
Horizontal_interpolation	Type of horizontal interpolation used. MM5 corresponds to MM5 coordinates and latlon to latitude/longitude coordinates.
Dot_coordinates	File containing coordinates of dot points. Used if Horizontal_interpolation is set to latlon.
	[meteo]
Relative_humidity_threshold	Minimum relative humidity above which cloud are formed.
Low_cloud_top_max	Low clouds maximum height (in m).
	Kz]

`Min_dry`	Minimum value of Kz in PBLH for dry conditions (in $m\,s^{-2}$).
`Min_wet`	Minimum value of Kz in PBLH for cloudy conditions (in $m\,s^{-2}$).
`Min_above_PBLH`	Minimum value of Kz above PBLH (in $m\,s^{-2}$).
`Max`	Maximum value for Kz (in $m\,s^{-2}$).

Among output files one may find:

- the altitude in meters,
- the air density (`AirDensity.bin`),
- the pressure in Pa (`Pressure.bin`),
- the temperature and temperature at 2 m in K (`Temperature.bin` and `Temperature_2m.bin`),
- the meridional wind, zonal wind, convective velocity and wind module at 10 m (`MeridionalWind.bin`, `ZonalWind.bin`, `ConvectiveVelocity.bin` and `WindModule_10m.bin`) in $m\,s^{-1}$,
- the boundary layer height in m (`PBLH.bin`),
- the vertical diffusion coefficients using Troen and Mahrt parameterization (`Kz.bin`) in $m^2\,s^{-1}$,
- the specific humidity in $kg\,kg^{-1}$ (`SpecificHumidity.bin`),
- the surface relative humidity (`SurfaceRelativeHumidity.bin`,
- the liquid water content in $kg\,kg^{-1}$ (`LiquidWaterContent.bin`),
- the cloud attenuation coefficients (`Attenuation.bin`),
- the soil moisture (`SoilMoisture.bin`),
- the aerodynamic resistance (`AerodynamicResistance.bin`),
- the friction velocity in $m\,s^{-1}$ (`FrictionModule.bin`).

3.4.7 Program `WRF-meteo`

Program `Polyphemus/preprocessing/meteo/WRF-meteo` processes WRF data and generates meteorological fields required by chemistry-transport models. Most fields are interpolated from WRF grid to a regular grid (latitude/longitude in the horizontal, altitudes in meters in the vertical).

Note that `WRF-meteo` needs as input data the land use cover which can be built using programs in `preprocessing/ground`.

Note for ECMWF users: program `WRF-meteo` is equivalent to what is performed by `meteo`, `attenuation` and `Kz` successively. Similarly to ECMWF files, `Kz_TM` can be used afterwards.

Program `WRF-meteo` can be launched as follows:

```
./WRF-meteo ../general.cfg WRF-meteo.cfg 2004-08-09_09-00-00
./WRF-meteo ../general.cfg WRF-meteo.cfg 2004-08-09_09-00-00 2004-08-10_09-00-00
./WRF-meteo ../general.cfg WRF-meteo.cfg 2004-08-09_09-00-00 1d
```

The configuration file `WRF-meteo.cfg` contains several options:

	`[paths]`
`Database_WRF_meteo`	Filename of the WRF input files. If `&D` appears in the file name, it is replaced by `YYYY-MM-DD` where `YYYY` is the year, `MM` the month and `DD` the day of the date given in command line.
`Directory_meteo`	Directory where output meteorological files are stored.
`File_Kz`	Name of the file where the vertical diffusion coefficients as computed with the Louis parameterization are stored.
	`[MM5]`
	`[accumulated_rain]`
`Prev_accumulated_rain`	Is the rain accumulated from the previous day?
	`[attenuation]`
`Type`	Parameterization to be used to compute cloud attenuation. Put `1` to use `RADM` parameterization or put `2` to use `ESQUIF` parameterization.
	`[clouds]`
`Min_height`	Minimum cloud basis height in m.
	`[Kz]`
`Min`	Lower threshold for vertical diffusion in $m^2\,s^{-1}$.
`Min_urban`	Lower threshold for vertical diffusion over urban areas, in $m^2\,s^{-1}$.
`Max`	Higher threshold for vertical diffusion in $m^2\,s^{-1}$.
`Apply_vert`	If set to `no`, the lower threshold is applied only at the top of the first layer, otherwise it is applied to all levels.

The program basically reads data in WRF output file and interpolates it in time and space to Polyphemus grid. WRF's files grids are described in its attributes and global variables (as every NetCDF file) and Polyphemus grid is described in `general.cfg`. Note that each time a field is loaded by `WRF-meteo`, all time steps are loaded in memory. Note that the fields are released from memory when unused, but you may still need a lot of memory for big WRF output files.

The program first computes the altitude of WRF layers, and converts the Polyphemus grid coordinates (latitude/longitude) to WRF grid coordinates (Lambert, Mercator or stereographic) for interpolations. Interpolations on the horizontal are performed in WRF grid for efficiency. The pressure is computed based on WRF fields. The winds are rotated: this gives meridional and zonal winds. The Richardson number is then computed (`ComputeRichardson`). The relative humidity and the critical relative humidity are computed respectively with `ComputeRelativeHumidity` and `ComputeCriticalRelativeHumidity`. The cloud fraction is computed with `ComputeCloudFraction`. For it the cloudiness and cloud height are diagnosed using `ComputeCloudiness` and `ComputeCloudHeight`. Finally attenuation coefficients are computed with `ComputeAttenuation_LWC` (RADM parameterization) or

`ComputeAttenuation_ESQUIF` (ESQUIF parameterization). The vertical diffusion coefficients are computed with `ComputeLouisKz` [Louis, 1979]. Finally photosynthetically active radiation are estimated, based on solar radiation and zenith angle (`ZenithAngle`).

Among output files one may find:

- the pressure and the surface pressure in Pa,
- the temperature, the surface temperature and the skin temperature in K,
- the meridional and zonal winds (`MeridionalWind.bin` and `ZonalWind.bin`) in $\mathrm{m\,s^{-1}}$,
- the Richardson number and the surface Richardson number,
- the boundary layer height in m,
- the vertical diffusion coefficients (time-dependent 3D field, defined on layer interfaces on the vertical, `Kz_Louis.bin`) in $\mathrm{m^2\,s^{-1}}$,
- the specific humidity in $\mathrm{kg\,kg^{-1}}$,
- the liquid water content in $\mathrm{kg\,kg^{-1}}$,
- the cloud attenuation coefficients (3D field, `Attenuation.bin`) in $[0, 2]$,
- the solar radiation intensity (`SolarRadiation.bin`) in $\mathrm{W\,m^{-2}}$,
- the rain intensity (`Rain.bin`) in $\mathrm{mm\,h^{-1}}$,
- the convective rain intensity (`ConvectiveRain.bin`) in $\mathrm{mm\,h^{-1}}$,
- the cloud height in m.

3.5 Deposition Velocities

Deposition velocities are generated on the basis of meteorological fields and land data. The programs must be launched after meteorological and ground preprocessing.

The computation of deposition velocities for Gaussian models is presented in Section 3.9.2.

3.5.1 Program `dep`

The program `dep` computes deposition velocities according to Wesely [1989] or Zhang et al. [2003].

In addition to `general.cfg`, the program reads the configuration in `dep.cfg`. In this file, paths to several files generated by programs `meteo`, `MM5-meteo` or `WRF-meteo` are given.

[paths]	
`SurfaceTemperature`	File where surface temperature is stored.
`SurfaceRichardson`	File where surface Richardson number is stored.
`SolarRadiation`	File where solar radiation is stored.
`WindModule`	File where wind module is stored.
`PAR`	File where photosynthetically active radiation is stored.
`PARdiff`	File where the diffuse part of the photosynthetically active radiation is stored.

PARdir	File where the direct beam part of photosynthetically active radiation is stored.
SpecificHumidity	File where (3D) specific humidity is stored.
SurfacePressure	File where surface pressure is stored.
FrictionVelocity	File where friction velocity is stored.
CanopyWetness	File where canopy wetness is stored.
Rain	File where rain is stored.
RoughnessHeight	File where roughness height is stored.
Type	Configuration file that describes land use cover (see below for details about this file).
ChemicalMechanism	Chemical mechanism used in your simulation. You can choose among racm, racm2 and cb05.
Data	File containing the data for species. This file should contain: the species name, the molecular weight ($g\,mol^{-1}$), Henry constant, diffusivity, reactivity, alpha [Zhang et al., 2003], beta [Zhang et al., 2003], Rm. An example for RACM/RADM is available in preprocessing/dep/input/species_data_racm.txt. Similar examples for RACM2 and CB05 mechanisms are also found.
Directory_dep	Directory where the output files are stored.
	[Species]
Ns	Number of species for which data are provided. This should be the number of columns in the file containing the data for species, for example, preprocessing/dep/input/species_data_racm.txt.
	[Options]
CellRoughness	If this option is set to yes, the roughness height used in calculations only depends on the model cell (and not on the land use category). In this case, it uses the data file whose path is given in entry RoughnessHeight (section [paths]). If the options is set to no (recommended), the roughness height depends on the land use category (see entry Type).
Ra	Parameterization used to compute the aerodynamic resistance. You can choose between fh (heat flux), fm (momentum flux) or diag (diagnostic).
Rb	Parameterization used to compute the quasi-laminar sublayer resistance. You can choose between friction and diag.
Rc	Parameterization used to compute the canopy resistance (Zhang et al. [2003] or Wesely [1989]).
Save_resistance	Should Ra, Rb and Rc be saved? This may take a lot of storage space: put no if you do not work on the deposition parameterizations.

Entry Type is the path to a configuration file whose entries should be:

File	Path to the file describing the land use cover. The number of categories in the file is deduced from its size, but it must be consistent with the data provided in the following entries (Midsummer, etc.)
Midsummer	Data file for midsummer (see below for details).

`Autumn`	Data file for autumn (see below for details).
`Late_autumn`	Data file for late autumn (see below for details).
`Snow`	Data file for snow (see below for details).
`Spring`	Data file for spring (see below for details).

The data files mentioned above for the five "seasons" must contain a column for each land use category with 22 parameters in each column. You may modify these files or create new files only if you are well aware of deposition parameterizations. With Polyphemus, a set of 5 files is provided for convenience, and any beginner should use them. They are suited for land use categories as defined in Zhang et al. [2002].

A key step is therefore to generate a land use description with these categories (referred as Zhang categories). The recommended program to generate this file is `luc-convert` which is described in Section 3.3.3. You should use this program to convert GLCF or USGS land cover to Zhang categories.

Please note that the program `dep` chooses which land use file to use according to *the month of the beginning date only*. Therefore, if you need deposition velocities for a date range during which the "season" changes, make sure to launch different simulations for the different seasons.

The program may be launched this way:

```
./dep ../general.cfg dep.cfg 2004-08-09 2d4h
```

3.5.2 Program `dep-emberson`

The program `dep-emberson` is used to compute deposition velocities for Castor model, using Emberson parameterization.

	`[paths]`
`Altitude`	File where altitude is stored.
`SurfaceTemperature`	File where surface temperature is stored.
`SurfaceRelativeHumidity`	File where surface relative humidity is stored.
`FrictionVelocity`	File where the friction velocity is stored.
`Attenuation`	File where the attenuation is stored.
`AerodynamicResistance`	File where the aerodynamic resistance is stored.
`LUC_file`	File containing the land use cover.
`Nc`	Number of land use cover categories.
`Nveg`	Number of vegetation classes.
`Land_data`	File containing land data in Chimere format.
`Species_data`	File containing the data for species (molecular weight, Henry constant, reactivity).
`Directory_dep`	Directory where the output files are stored.
	`[Species]`
`Ns`	Number of species for which data are provided.

The program must be launched with:

```
./dep-emberson ../general.cfg dep-emberson.cfg 2004-08-09 2004-08-12
```

3.6 Emissions

Emissions are generated on the basis of land data (anthropogenic emissions) and meteorological fields (biogenic emissions). The programs must be launched after meteorological and ground preprocessing.

For Gaussian models, a preprocessing step may also be required in case line emissions are included (see Section 3.9.1).

3.6.1 Mapping Two Vertical Distributions: `distribution`

Program `distribution` may be used to define the distribution of emissions along the vertical. It reads the vertical distribution of emissions in some input grid and maps this distribution on an output vertical grid. Thus it generates a file with the vertical distribution of emissions in the output grid. It is based on AtmoData function `ComputeVerticalDistribution`.

Running this program is not compulsory. Even if the vertical distribution of emissions is required to compute anthropogenic emissions (program `emissions`), the vertical distribution can be generated by other means (including "by hand").

In addition to the domain definition (Section 3.2.2), program `distribution` reads a configuration file such as `emissions.cfg`:

	`[domain]`
`Nz`	Number of output vertical levels.
`Vertical_levels`	Path to the text file that stores the altitudes (in m) of output level interfaces (hence `Nz+1` values are read).
	`[EMEP]`
`Nz_in`	Number of input vertical levels.
`Vertical_levels`	Path to the text file that stores the altitudes (in m) of input level interfaces (hence `Nz_in+1` values are read).
`Vertical_distribution`	Path to the file with the input vertical distribution of emissions. This file should contain one line per emission sector. Each line contains the percentage of emissions at ground level (first column) and the percentage of emissions in each vertical level (`Nz_in` following columns).
`Polair_vertical_distribution`	Path to the output file where the output vertical distribution of emissions should be stored. The format is the same as in file `Vertical_distribution`.
`Nsectors`	Number of activity sectors.

The program must be launched with:

```
./distribution ../general.cfg emissions.cfg
```

3.6.2 Anthropogenic Emissions (EMEP): `emissions`

Program `emissions` processes an EMEP emission inventory and generates (anthropogenic) surface and volume emissions needed by Castor or Polair3D.

First you must download the "emissions used in EMEP models" from http://www.ceip.at/emission-data-webdab/emissions-used-in-emep-models/. Download emissions for CO, NH3, NMVOC, NOx, SOx, PM2.5 and PMcoarse and make sure to have a file for each pollutant

called `CO.dat`, `NH3.dat`, `NMVOC.dat`, `NOX.dat`, `SOX.dat`, `PM2.5.dat` and `PMcoarse.dat`.

Download the files with the following options:

- for all countries ("ALL");
- for the year of your choice;
- for all activity sectors (SNAP): "All Sectors"; note that emissions for the eleventh sector will be ignored by `emissions` – they are better estimated with program `bio` (see Section 3.6.3);
- a single pollutant: you must download the emissions for one pollutant at a time so that you should have one file for each pollutant (`CO.dat`, `NH3.dat`, ...);
- in format "Grid (50 km × 50 km), Semicolon-Separated";
- whatever for entries "HTML Table x-Axis" and "HTML Table y-Axis";
- with the footnotes (recommended but not mandatory).

Then click on "Show Data" and save the data in the right file (`NOX.dat`, `SOX.dat`, ...). The EMEP website is subject to changes, so if the explanations provided above are not consistent with EMEP website, you may contact Polyphemus team (`polyphemus-help@lists.gforge.inria.fr`).

In addition to the domain definition (Section 3.2.2), program `emissions` reads a configuration file such as `emissions.cfg`:

	`[paths]`
`Directory_surface_emissions`	Directory where the computed surface emissions are stored.
`Directory_volume_emissions`	Directory where the computed volume emissions are stored. This should be different from `Directory_surface_emissions` since files for surface emissions and volume emissions have the same names (species names).
	`[options]`
`Chemical_mechanism`	Chemical mechanism used in your simulation. You can choose among `racm`, `racm2`, `cb05` and `cb05-siream`.
`Molecular_weights`	File containing molecular weights of species.
	`[EMEP]`
`Polair_vertical_distribution`	File where the vertical distribution of emissions is stored. This file should contain one line per emission sector. Each line contains the percentage of emissions at ground level (first column) and the percentage of emissions in each vertical level (`Nz` following columns).
`Input_directory`	Directory containing EMEP emissions inventory.
`Hourly_factors`	File defining hourly factors (see below).
`Weekdays_factors`	File defining weekdays factors (see below).

`Monthly_factors`	File defining monthly factors (see below).
`Time_zones`	File defining the time zone for various countries.
`Nx_emep`	Number of cells along longitude (integer) in EMEP grid.
`Ny_emep`	Number of cells along latitude (integer) in EMEP grid.
`Ncountries`	Maximum code number of the countries covered by the inventory. If a code number in the inventory is greater than or equal to `Ncountries`, an error message is thrown.
`Species`	Names of inventory species.
`Nsectors`	Number of activity sectors.
`Urban_ratio`	Emission ratio for urban areas (see below).
`Forest_ratio`	Emission ratio for forest (see below).
`Other_ratio`	Emission ratio for other areas (see below).
	`[LUC]`
`File`	Path to land use cover file.
`x_min`	Longitude in degrees of the center of the lower-left cell in LUC grid.
`Delta_x`	Step length (in degrees) along longitude of LUC grid.
`Nx`	Number of cells along longitude (integer) in LUC grid.
`y_min`	Latitude in degrees of the center of the lower-left cell in LUC grid.
`Delta_y`	Step length (in degrees) along latitude of LUC grid.
`Ny`	Number of cells along latitude (integer) in LUC grid.
	`[Species]`
`N`	Maximum number of emitted species (see below).
`Aggregation`	Aggregation matrix file (relations of the emitted species to the real chemical species).
`Speciation_directory`	Directory in which, for each inventory species `XXX`, a file `XXX.dat` contains the speciation to real chemical species as function of the emission sector (columns).
`Deposition_factor_NH3`	Part of emitted NH_3 which is deposited right away.

The program `emissions` reads EMEP emissions inventory, multiplies them by temporal factors and interpolates them on Polair3D grids. EMEP emissions are read with AtmoData function `ReadEmep`. The spatial interpolation is performed with `EmepToLatLon`.

LUC file This file gives the land categories in GLCF description in a *domain that must contain your simulation domain* – but this LUC domain should not be too large because of computational costs. You can generate this file using program `extract-glcf` (see Section 3.3.5).

Urban, forest and "other" ratios Ratios `Urban_ratio`, `Forest_ratio` and `Other_ratio` enable to distribute emissions of an EMEP cell according to the type of land (urban, forests and other categories). For instance, in an EMEP cell, emissions are distributed so that the ratio between total urban emissions and total emissions is `Urban_ratio` on top of the sum of `Urban_ratio`, `Forest_ratio` and `Other_ratio`.

Temporal Factors EMEP emissions are provided as annual values. They are multiplied by temporal factors to estimate their time evolution (as function of month, week day and hour) in

all emission sectors and in all countries.

Here are examples on how these factors should be provided:

- `monthly_factors.dat` gives the factors for each country index (`CC`), each activity sector (`SNAPsector`) and each month.

```
# Formate: CC SNAPsector  JAN    FEB    MAR    APR    MAY    JUN    JUL  ...
2  1  1.640  1.520  1.236  1.137  0.798  0.459  0.393 ..
```

- `weekdays_factors.dat` gives the factors for each country index (`CC`), each activity sector (`SNAPsector`) and each day of the week.

```
# Formate: CC SNAPsector  MON      TUE      WED      THU   ...
2  1  1.0159   1.0348   1.0399   1.0299   ...
```

- `hourly_factors.dat` gives the factor for each activity sector (`SNAPsector`) and each hour. The hours are local time and they must range from 1 to 24.

```
#  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  ...
1  1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 ...
```

In addition to these factors a file called `time_zones.dat` is necessary. It gives the various countries in EMEP inventories and their time zone offset to GMT. Please note that the list of countries in the inventories may vary without warnings. If it happens, the code should raise an error and tell you which country code in `time_zones.dat` is unknown.

Number of emitted species `N` should be greater or equal to the sum of species number of each type of emissions. There are seven types in EMEP inventory: `CO`, `NMVOC`, `NOX`, `SOX`, `NH3`, `PM2.5` and `PMcoarse`.

Chemical mechanism	Total	CO	NMVOC	NOX	SOX	NH3	PM2.5	PMcoarse
RACM	**42**	1	14	3	2	1	19	2
RACM2	**65**	1	37	3	2	1	19	2
CB05	**43**	1	15	3	2	1	19	2

The program automatically takes into account summer and winter time. The output emissions start at 00:00 UTC.

3.6.3 Biogenic Emissions for Polair3D Models: `bio`

Program `bio` computes biogenic emissions on the basis of meteorological fields and land use cover.

In addition to the domain definition (Section 3.2.2), below is the information required in the configuration for `bio` (see example `bio.cfg`):

`[paths]`	
`SurfaceTemperature`	Binary file where the surface temperature is stored.
`PAR`	Binary file where the photosynthetically active radiation is stored.
`LUC_file`	Binary file where the land use cover is stored.

`Land_data`	Data file giving emission factors for isoprene, terpenes and NO_x for all land categories defined in `LUC_file`. In this file, each line (which is not empty or does not start with "#") provides data for one land use category. For such a line, the first 55 characters are discarded: you may put the category number and description for convenience. Then four columns are read with the biomass density ($g\,m^{-2}$), and the emission factors for isoprene, terpenes and NO_x (in this order). Two examples are provided with `land_data_glcf.dat` and `land_data_usgs.dat`, to be used in combination with land use cover generated by `luc-glcf` or `luc-usgs` respectively.
`Directory_bio`	Directory where output biogenic emissions are stored.
	`[biogenic]`
`Delta_t`	Time step (in hours) for the output biogenic emissions. For simulations with Polair3D, anthropogenic emissions and biogenic emissions must have the same time step (that is, usually one hour).
`Rates`	Should emission rates be saved? These rates are not needed by chemistry-transport models.
`Terpenes`	Names of the species included in terpenes emissions. For RACM Stockwell et al. [1997] and RACM2 Goliff and Stockwell [2008], put `API` and `LIM`. For CB05 Yarwood et al. [2005], put `TERP`.
`Terpenes_ratios`	Distribution of terpenes emissions among species (entry `Terpenes`). For RACM and RACM2, put 0.67 0.33 (for `API` and `LIM`, respectively). For CB05, put 1.0 (for `TERP`).

Biogenic emissions are computed according to Simpson et al. [1999]. Meteorological data is first interpolated in time so that its time step is `Delta_t` (section `[biogenic]`). Emission rates are then computed using AtmoData function `ComputeBiogenicRates` and emissions using `ComputeBiogenicEmissions`.

3.6.4 Biogenic Emissions for Castor Models: `bio-castor`

Program `bio-castor` is slightly different from `bio`, in particular regarding the data provided. Its configuration file, e.g. `Polyphemus/preprocessing/bio/bio-castor.cfg` defines:

	`[paths]`
`SurfaceTemperature`	Binary file where the surface temperature is stored.
`WindModule_10m`	Binary file where the wind module at 10 m is stored.
`Attenuation`	Binary file where attenuation data are stored.
`SoilMoisture`	Binary file where the moisture of the ground is stored.
`ConvectiveVelocity`	Binary file where the convective velocity is stored.
`Land_data`	Land data in Chimere format.
`Directory_bio`	Directory where output biogenic emissions are stored.
	`[biogenic]`
`Minimum_wind_velocity`	Minimum value of `WindModule_10m`.
`Terpenes`	Species between which terpene emissions are distributed.
`Terpenes_ratios`	Ratio of the terpene emissions for each of the above species.

3.6.5 Sea Salt Emissions: `sea-salt`

Program `sea_salt` computes the emissions of sea-salt aerosols. Its options and parameters are given in `sea_salt.cfg`.

	`[paths]`
`Surface_wind_module_file`	Binary where the wind module at surface is stored.
`Directory_sea_salt`	Directory where sea-salt emissions are stored.
	`[sea_salt]`
`Parameterization`	Parameterization used for sea salt emissions: Monahan (Monahan et al. [1986]) or Smith (Smith and Harrison [1998]).
`Threshold_radius`	Radius above which the parameterization is used (in µm).
`Delta_t`	Time step for sea-salt emissions computation.
	`[LUC]`
`File`	File containing land use cover.
`Nb_luc`	Number of land categories.
`Sea_index`	Index of sea in land categories. It is 0 for GLCF description and 15 for USGS description.
	`[PM]`
`Section_computed`	Should diameter classes bounds be computed? Otherwise they are read in `File_sections`.
`Diameter_min`	Minimum diameter if diameter classes bounds are computed.
`Diameter_max`	Maximum diameter if diameter classes bounds are computed.
`Nsections`	Number of diameter classes.
`File_sections`	File containing the diameter classes bounds if they are not computed.
	`[sea-salt_composition]`
`NA, CL, SO4`	Fraction of NA, CL and SO_4 in sea salt.

3.7 Initial Conditions: `ic`

Climatological concentrations from Mozart 2 [Horowitz et al., 2003] are used to generate initial concentrations for photochemistry simulation with Polair3D.

Program `ic` has been tested with Mozart 2 output files downloaded on NCAR data portal at `http://cdp.ucar.edu`.

To download any data from the NCAR Community Data Portal you need to register. This is quite easy and fast but there is a second step. You also have to ask for an access to Mozart data specifically. This takes longer as the application has to be reviewed by someone but it should go without problems if you say that you need Mozart data to generate initial and boundary conditions for a CTM.

The data can be found in "ACD: Atmospheric Chemistry Models, Data Set and Visualization Tools". If you registered, in this section of the site, you should be able to access "MOZART (Model for OZone And Related chemical Tracers)". This opens a page with various informations about Mozart and in particular, in "Nested Collections" a link named "MOZART-2 MACCM3 Standard Simulation (v2.1)". Click on this link.

Currently, the direct link is http://cdp.ucar.edu/browse/browse.htm?uri=http://dataportal.ucar.edu/metadata/acd/mozart/mozart2/mozart_v2_1_maccm3.thredds.xml, but if this changes a search for "MOZART MACCM3" should lead you to the right page.

There are 38 files available in NCAR Community Data Portal (from hc0040.nc to hc0077.nc). One file gathers data for ten consecutive days. File hc0040.nc start at 26 December. Data in those files have been generated for a typical year, which means they can be use to generate initial conditions for any year.

In addition to the domain definition (Section 3.2.2), below is the information required in the configuration for `ic` (see example `ic.cfg`):

	`[ic_input_domain]`
`Date_ic`	Date for which initial conditions are generated.
`Nt`	Number of time steps in Mozart 2 files (integer).
`Delta_t`	Time step of Mozart 2 files (in hours).
`Nx`	Number of grid points along latitude in Mozart 2 files (integer).
`Ny`	Number of grid points along longitude in Mozart 2 files (integer).
`Nz`	Number of vertical levels in Mozart 2 files (integer).
`Database_ic`	Directory where the Mozart 2 files are available. Mozart 2 filenames are in form `h00xx.nc` where `xx` is computed by the program according to the date `Date_ic`.
	`[ic_files]`
`Chemical_mechanism`	Chemical mechanism used in your simulation.
`Species`	File providing correspondence between the name of species in Mozart 2 files and the name of species in simulation. In this file, the first column contains Mozart 2 species. After each Mozart 2 species name, the corresponding output species (e.g., RACM species) is put, if any. If Mozart 2 species gathers two output species, put the names of all output species followed by their proportion in Mozart 2 bulk species. For instance, the line `C4H10 HC5 0.4 HC8 0.6` splits Mozart 2 species `C4H10` into `HC5` (40%) and `HC8` (60%). Three examples are provided: `preprocessing/bc/species_racm.dat`, `species_racm2.dat` and `species_cb05.dat`. Two other examples are also provided for RACM: `species_racm_v1.dat` and `species_racm_v2.dat`
`Molecular_weights`	File providing the molecular weights of output species.
`Directory_ic`	Directory where the output initial conditions must be stored.

The name of the Mozart 2 files must be in the form `h00xx.nc` where `xx` is computed as shown in Equation 3.1.

$$xx = 40 + int\left[\frac{N_d + 6}{10}\right] \tag{3.1}$$

with N_d the number of days since the beginning of the year (0 for first January) and $int(x)$ represents the integral part of x.

The program `Polyphemus/preprocessing/ic` automatically select the file to use with the date given in `ic.cfg`.

Output results are in $\mu g\,m^{-3}$.

In case your Mozart 2 files do not satisfy this format (this may happen if Mozart files are updated on the NCAR data portal), you may modify the code or contact Polyphemus team at

polyphemus-help@lists.gforge.inria.fr.

3.8 Boundary Conditions

3.8.1 Boundary Conditions for Polair3D

Program `bc`

Boundary condition for gaseous species are generated using Mozart 2 files. See Section 3.7 on how to get those files.

In addition to the domain definition (Section 3.2.2), below is the information required in the configuration for `bc` (see example `bc.cfg`):

	`[bc_input_domain]`
`Nt`	Number of time steps in Mozart 2 files.
`Delta_t`	Time step of Mozart 2 files (in hours).
`Nx`	Number of grid points along longitude in Mozart 2 files (integer).
`Ny`	Number of grid points along latitude in Mozart 2 files (integer).
`Nz`	Number of vertical levels in Mozart 2 files (integer).
	`[bc_files]`
`Directory_bc`	Directory where the output boundary conditions must be stored.
`Species`	File providing correspondence between the name of species in Mozart 2 files and the name of species in simulation. In this file, the first column contains Mozart 2 species. After each Mozart 2 species name, the corresponding output species (e.g., RACM species) is put, if any. If Mozart 2 species gathers two output species, put the names of all output species followed by their proportion in Mozart 2 bulk species. For instance, the line `C4H10 HC5 0.4 HC8 0.6` splits Mozart 2 species `C4H10` into `HC5` (40%) and `HC8` (60%). Three examples are provided: `preprocessing/bc/species_racm.dat`, `species_racm2.dat` and `species_cb05.dat`. Two other examples are also provided for RACM: `species_racm_v1.dat` and `species_racm_v2.dat`
`Molecular_weights`	File providing the molecular weights of output species.

Program `bc` processes an entire Mozart 2 output file. If this file contains concentrations for 10 days, the program generates boundary conditions for 10 days with a time-step of 24 hours.

The program must be launched with:

```
./bc ../general.cfg bc.cfg /net/libre/adjoint/mallet/mozart/h0067.nc
```

The last argument is the path to the Mozart 2 file. You have to select the file to use according to date as shown in Equation 3.1.

The results are in $\mu g\,m^{-3}$. They are stored as `&f_&c.bin` where `&f` is replaced by the name of the species and `&c` by the direction associated with the boundary condition (x, y or z). For example, the concentrations in `O3_x.bin` are interpolated at both ends of the domain along x, for all grid points along y and z.

Program `bc-dates`

Program `bc-dates` is very similar to `bc`. The main difference is that, instead of a Mozart file, two dates are given in command line. The program computes boundary conditions for all Mozart files between the two dates, successfully managing year changes.

The configuration file for program `bc-dates` (see example `bc-dates.cfg`) is essentially the same as the one for `bc`, with one addition:

	`[bc_files]`
`Directory_mozart`	Directory where the Mozart 2 files are available. Mozart 2 filenames are in form `h00xx.nc` where `xx` is computed by the program according to the date (see Equation 3.1).

The program must be launched with:

```
./bc-dates ../general.cfg bc-dates.cfg 2004-07-30 2004-08-12
```

3.8.2 Boundary Conditions for Castor: `bc-inca`

Boundary condition for Castor models are generated using INCA files.

In addition to the domain definition (Section 3.2.2), below is the information required in the configuration for `bc-inca` (see example `bc-inca.cfg`):

	`[bc_input_domain]`
`x_min`	Abscissa of the center of the lower-left cell of INCA grid, in degrees (longitude).
`Delta_x`	Step length along x in INCA files, in degrees (longitude).
`Nx`	Number of grid points along longitude in INCA files (integer).
`y_min`	Ordinate of the center of the lower-left cell of INCA grid, in degrees (latitude).
`Delta_y`	Step length along y in INCA files, in degrees (latitude).
`Ny`	Number of grid points along latitude in INCA files (integer).
`Nz`	Number of vertical levels in INCA files (integer).
`Ns`	Number of species in the INCA file.
`Species`	File giving the species in the INCA file.
	`[bc_files]`
`Nt`	Number of time steps necessary for the output (in hours).
`Directory_bc`	Directory where the output boundary conditions must be stored.

Program `bc-inca` reads the INCA file (which is a text file) and saves its results for the number of time step given in the configuration file (`Nt`).

The program must be launched with:

```
./bc-inca ../general.cfg bc-inca.cfg /u/cergrene/a/ahmed-dm/A/raw_data/INCA/INCA.07
```

The last argument is the path to the INCA file. The number at the end of the file name represents the month.

The results are stored as `&f_&c.bin` where `&f` is replaced by the name of the species and `&c` by the direction associated with the boundary condition (x, y or z).

Note that initial conditions for Castor are interpolated from the boundary conditions and do not need to be computed separately.

3.8.3 Boundary Conditions for Aerosol Species: `bc-gocart`

Boundary conditions for aerosol species are obtained using Gocart model[†1] thanks to the program `bc-gocart`.

Gocart format and conventions

Gocart model usually provides files with the following naming convention:

file name	signification
yyyymm.XX.vs.g	6-hourly concentrations in $\mathrm{g\,m^{-3}}$.
yyyymm.XX.vs.g.day	daily averaged concentrations in $\mathrm{g\,m^{-3}}$.
yyyymm.XX.vs.g.avg	monthly averaged concentrations in $\mathrm{g\,m^{-3}}$.

where *yyyymm* is the year and month (e.g., 200103), *XX* is the Gocart species, which can be either SU (sulfur), CC (carbonaceous), DU (dust), SS (sea-salt), and *vs* is the version (e.g., STD.tv12).

Gocart species may have further speciations:

- **SU** (sulfur): Total 4, 1-DMS, 2-SO2, 3-SO4, 4-MSA.
- **CC** (BC+OC): Total 4, 1-hydrophobic BC, 2-hydrophobic OC, 3-hydrophilic BC, 4-hydrophilic OC.
- **DU** (dust): Total 5, 1-Re=0.1-1, 2-Re=1-1.8, 3-Re=1.8-3, 4-Re=3-6, 5-Re=6-10 µm. The first group (0.1-1 µm) contains the following subgroups:
  - 0.10-0.18 µm (fraction = 0.01053)
  - 0.18-0.30 µm (fraction = 0.08421)
  - 0.30-0.60 µm (fraction = 0.25263)
  - 0.60-1.00 µm (fraction = 0.65263)
- **SS** (sea-salt): Total 4, 1-Re=0.1-0.5, 2-Re=0.5-1.5, 3-Re=1.5-5, 4-Re=5-10 µm.

The data format of Gocart files is "direct access binary, 32 bits, big endian". As an example, here is how they should be read in Fortran 77 language:

```
dimension Q(imx,jmx,lmx)
do k=1,nstep
 do n=1,nmx
  read(unit) nt1,nt2,nn,Q
 end do
enddo
```

where

- imx = total number of longitudinal grid (144),

[†1] http://code916.gsfc.nasa.gov/People/Chin/gocartinfo.html

- jmx = total number of latitudinal grid (91),
- lmx = total number of vertical layers (version dependent),
- nmx = total number of species (4 or 5, see species list above),
- nt1 = yyyymmdd after 2000 (year-month-day, e.g., 20010201), or yymmdd before 2000 (e.g., 970101)
- nt2 = hhmmss (hour-minute-second, e.g., 120000)
- nn = tracer number (see species list above)
- Q = 3-dimensional concentration of tracer nn
- nstep = total time step (e.g., in 200101, nstep=4*31 for 4-times/day, nstep=31 for daily average files, and nstep=1 for monthly average files).

Important

- If you plan to read Gocart data on your own, do not forget to translate files from big endian to little endian if necessary.
- The conventions and format of Gocart files may change in the future.

Fields resolution

The horizontal resolution of Gocart fields is 2 degree latitude $\times$ 2.5 degree longitude, except at the poles where latitudinal resolution is 1 degree. In other words the longitude interval is $[-180 : 2.5 : 177.5]$ (144 cells) and the latitude one is $[-89.5 - 88 : 2 : 8889.5]$ (91 cells).

The vertical resolution is given as a given number of vertical sigma levels. The number of vertical levels depends of the year :

- **1980-1995**: 20 sigma layers centered at 0.993936, 0.971300, 0.929925, 0.874137, 0.807833, 0.734480, 0.657114, 0.578390, 0.500500, 0.424750, 0.352000, 0.283750, 0.222750, 0.172150, 0.132200, 0.100050, 0.0730000, 0.0449750, 0.029000, 0.00950000
- **1996-1997**: 26 vertical sigma layers centered at 0.993935, 0.971300, 0.929925, 0.875060, 0.812500, 0.745000, 0.674500, 0.604500, 0.536500, 0.471500, 0.410000, 0.352500, 0.301500, 0.257977, 0.220273, 0.187044, 0.157881, 0.132807, 0.111722, 0.0940350, 0.0792325, 0.0668725, 0.0565740, 0.0447940, 0.0288250, 0.00997900
- **2000-2002**: 30 vertical sigma layers centered at 0.998547, 0.994147, 0.986350, 0.974300, 0.956950, 0.933150, 0.901750, 0.861500, 0.811000, 0.750600, 0.682900, 0.610850, 0.537050, 0.463900, 0.393650, 0.328275, 0.269500, 0.218295, 0.174820, 0.138840, 0.109790, 0.0866900, 0.0684150, 0.0539800, 0.0425750, 0.0335700, 0.0239900, 0.0136775, 0.00501750, 0.00053000

Gocart files processing

Gocart files are handled by `bc-gocart` program which takes 5 arguments:

```
./bc-gocart ../general.cfg bc-gocart-CC.cfg 200101.CC.STD.tv15.g.day 200101 7
```

where

- `../general.cfg` is the general configuration file,
- `bc-gocart-CC.cfg` is the configuration file for CC Gocart species,
- `200101.CC.STD.tv15.g.day` is the Gocart file,
- `200101` is the date of Gocart file, this file corresponds to daily carbonaceous values during month of January 2001,
- `7` is the number of days for which boundary conditions are computed.

The configuration file `bc-gocart-CC.cfg` provides all necessary informations to read Gocart fields and to translate them into `polair3d` species.

	`[paths]`
`Directory_bc`	Directory where output will be written.
	`[bc_input_domain]`
`x_min`	Minimum longitude in Gocart resolution.
`y_min`	Minimum latitude in Gocart resolution.
`Delta_x`	Gocart longitude resolution.
`Delta_y`	Gocart latitude resolution.
`Nx`	Number of grid cells in the longitude Gocart axe.
`Ny`	Number of grid cells in the latitude Gocart axe.
`Nz`	Number of Gocart vertical layers.
`Sigma_levels`	File where are written the center of Gocart sigma levels.
`Scale_height`	Scale height in meter.
`Surface_pressure`	Surface pressure in atm.
`Top_pressure`	Pressure at top of Gocart level (in atm).

There are two more sections in configuration file.

The first one is `[input_species]`. Each non blank line of this section corresponds to one speciation of Gocart species, e.g. CC is sub-divided in CC-1, CC-2, CC-3, CC-4. The range after the delimiter ":" is the aerosol size range (in SIμm) to which this sub-species apply. Most of the time this is the whole aerosol size range of `polair3d` model (e.g. $0.1 - 10.0$), but in the case of dust (DU) each sub-species may correspond to a precise part of the `polair3d` aerosol size range, see configuration file `bc-gocart-DU.cfg` for an illustration.

The second section is `[output_species]`. Each non blank line of this section corresponds to one aerosol species of `polair3d` model. The columns after ":" delimiter correspond to the Gocart sub-species. Therefore the number of line in previous section must equal the number of column after ":" delimiter. The numbers in these columns are the fraction (between $0.0 - 1.0$) of given Gocart sub-species that will contribute to given model species. As an example in `bc-gocart-CC.cfg` the first line

```
PBC:    1.      0.      1.      0.
```

means that sub-species CC-1 and CC-3 will fall into PBC `Polair3D` species, and nowhere else. In the same way the following line

```
PPOA:   0.      0.4     0.      0.4
```

means that PPOA species is composed of 40% of CC-2 and 40% of CC-4.

Important The Gocart files are proceeded month by month.

- The beginning date of computation is the one provided in `../general.cfg` if the beginning month is equal to the Gocart month, the beginning of Gocart month otherwise.
- An end date is deduced from the number of days given in argument. If this end date is after the end of Gocart month, the end date is set to the end of Gocart month.
- If some boundary files already exist, the program `bc-gocart` will not overwrite them but append its result to each.

For example if you want to compute boundary conditions between 15th of April to 15th of June 2001, you would launch `bc-gocart` three times:

```
./bc-gocart ../general.cfg bc-gocart-CC.cfg 200104.CC.STD.tv15.g.day 200104 15
./bc-gocart ../general.cfg bc-gocart-CC.cfg 200105.CC.STD.tv15.g.day 200105 31
./bc-gocart ../general.cfg bc-gocart-CC.cfg 200106.CC.STD.tv15.g.day 200106 15
```

The python script `bc-gocart.py` provides a much easier way to compute boundary conditions. In particular, you do need to worry about how many times to launch `bc-gocart`. In the last example, one should simply launch:

```
./bc-gocart.py ../general.cfg 2001-04-15 61
```

where `2001-04-15` is the beginning date of the simulation (it **must** be the same as in `../general.cfg`) and `61` is the number of days to

process. Pay attention that some paths must be supplied inside this script (paths of Gocart configuration and data files) for it to work.

You are strongly advised to use script `bc-gocart.py`, as the computation of Gocart months to use and the number of days to use for each month is performed automatically.

Remark

Gocart **does not** provide any boundary conditions for nitrate and ammonia, you have to compute them on your own. Nevertheless a quick way to compute boundary conditions for ammonia is to apply electroneutrality to already computed aerosol boundary conditions from Gocart (or whatever else in fact). This can be done by `bc-nh4` program which takes two arguments:

```
./bc-nh4    ../general.cfg bc-nh4.cfg
```

The electroneutrality equation is set in configuration file `bc-nh4.cfg`.

You can also compute it with script `bc-gocart.py` using the option `--nh4`:

```
./bc-gocart.py --nh4 ../general.cfg 2001-04-15 61
```

3.9 Preprocessing for Gaussian Models

3.9.1 Program `discretization`

The aim of this program is to discretize a line emission in the case of a continuous source (plume source) or an instantaneous one (puff source). It reads a line source given by two points or more, and gives in return the discretized source. The output data is a list of point sources whose coordinates have been calculated given the line coordinates and the number of points to discretize the line, or the source velocity in the case of a moving source (for puff sources only).

The program `discretization` is launched with one configuration file. The reference configuration file is `discretization.cfg`. It contains the following information:

	`[trajectory]`
`Trajectory_file`	Path to the data file that contains the line coordinates.
`Np`	Number of points per segment used to discretize the trajectory. Used only when the source is not moving.
`Delta_t`	Time step to calculate the discretized trajectory in the case of a moving source.
	`[source]`
`Source_type`	Source type: `puff` or `continuous`. If the type is `puff`, section `[puff-source]` is read. Section `[plume-source]` is used when the type is `continuous`.
`Species`	List of names of the species emitted by the source.
`Velocity`	Velocity of the gas or aerosol emitted by the source (in $\mathrm{m\,s^{-1}}$).
`Temperature`	Temperature of the gas or aerosol emitted by the source (Celsius degrees).
`Diameter`	Diameter of the source in m.
`Date_beg`	Beginning date of the emission.
	`[plume-source]`
`Rate`	Source rate per unit of length (in mass/s/m) of the line source (one per species).
`Date_end`	Ending date of the emission.
	`[puff-source]`
`Quantity`	Total mass per unit of length (in mass/m) released on the line source (one per species).
`Source_velocity`	Source velocity (in $\mathrm{km\,h^{-1}}$) (0. for non mobile sources).
	`[output]`
`With_comment`	Are comments written?
`Source_file`	Path to the data file where the list of sources will be written.

The associated data file (reference: `line-emission.dat`) contains the coordinates of line sources to be discretized. Each line corresponds to one segment, defined by its extremity points. The data file contains eight columns corresponding to the data for each of the two extremity points (nodes) of a segment: for each node, a unique identifying number (integer) is given, as well as three cartesian coordinates (meters). Thus, the columns are labeled Id1, X1, Y1, Z1, (first node) and Id2, X2, Y2, Z2 (second node). If several segments share the same node, the same index has to be given to the node every time it appears in the file, so that it can be easily identified and taken into account only once by the program.

This is an example of data file, defining a straight line emission:

```
#Id1 X1(m) Y1(m) Z1(m) Id2 X2(m) Y2(m) Z2(m)
1 100. 90. 1. 2 100. 110. 1.
```

Alternatively, the line source file can define one continuous line made of segments (broken line). In this case, each line contains the coordinates of the broken line nodes (X, Y, Z in meters),

and the line extremities are given by the first and last nodes in the file. The file must contain at least the coordinates of two points. There is no identifying number here, since each node is assumed to appear only once in the file. Please note that only one way of describing line sources must be used within the data file : either the list of segments (eight columns for all lines in the file), or the broken line (three columns for all lines in the file).

This is an example of data file, defining a straight line emission between two points:

```
#X(m)   Y(m)    Z(m)
0.      0.     30.
20.     0.     30.
```

The output data file contains a list of point sources. It is presented as a list of sections named `[source]`, each section containing the coordinates and other data for one point source. All points coordinates have been calculated by the program. The emission rate is the same for all points except at both extremities of each line source. Indeed, to correct side effects the emission rate is divided by two at extremities but the total emission rate on each line source remain unchanged.

The program `discretization_aer` is the same, except that the output is a point emission file formatted for aerosol species to be used with GaussianPlume_aer or GaussianPuff_aer. In `discretization.cfg`, the name `Species` is replaced by `Species_name`, there can be only one species and one bin per source, and the rate is in mass/s instead of mass/s/m.

3.9.2 Programs `gaussian-deposition` and `gaussian-deposition_aer`

The aim of these programs is to calculate the scavenging coefficient and the deposition velocity of the species. The program `gaussian-deposition` is used when all species are gaseous species, and `gaussian-deposition_aer` is used when some or all species are aerosol species. The input data are meteorological data and species data, and the output file is a file containing meteorological data and the scavenging coefficients and deposition velocities of all species. This file can be used as input meteorological file for the programs `plume` and `puff` for gaseous species, or `plume_aer` and `puff_aer` in the case of aerosol species.

Program gaussian-deposition

Configuration File The program `gaussian-deposition` is launched with one configuration file and two input files. The configuration file contains the path to the two input files and to the output file. The reference configuration file is `gaussian-deposition.cfg`. It contains the following information:

	`[data]`
`Species`	Path to the data file that contains the species data.
`Meteo`	Path to the data file that contains the meteorological data.
	`[scavenging]`
`Type`	Parameterization to be used to calculate the scavenging coefficients.
`File`	Path to the file that contains the scavenging coefficients for all species. Read only if `Type` is set to `file`.
	`[deposition]`

`Type`	Parameterization to be used to calculate the deposition velocities.
`File`	Path to the file that contains the scavenging coefficients for all species. Read only if `Type` is set to `file`.
	`[output]`
`With_comment`	Are comments written in the output file? (put yes or no).
`Output_file`	Path to the file where the output data are written.

The parameterization type for the scavenging coefficient can be chosen between:

- `none`: the scavenging coefficient is set to 0. for all species,
- `constant`: the scavenging coefficient is constant for one given species and entered in the species file,
- `file`: the scavenging coefficients are read in a file, for each species and meteorological situation,
- `belot`: the scavenging coefficient is calculated with a Belot parameterization. In that case, the input data are a rainfall rate given in the meteorological data file, and coefficients a and b given for each species in the species file.

Concerning the deposition velocity, the type can be chosen between:

- `none`: the deposition velocity is set to 0. for all species,
- `file`: the deposition velocities are read in a file, for each species and meteorological situation,
- `constant`: the deposition velocity is constant for one given species and is given in the species file.

Input Files There are two input data files for this program: the meteorological data file (reference: `meteo.dat`) and the species file (reference: `species.dat`).

1. Meteorological data file: it contains as many sections as there are meteorological situations. For each situation, meteorological data (temperature, wind ...) are given as described in section 5.1.5. They will be written unchanged in the output file which will be the meteorological data file of the main program. Other meteorological data might be needed, depending on the chosen parameterization to compute scavenging coefficients and deposition velocities. Currently, the only parameterization that needs other information is the Belot parameterization. If the type `belot` is chosen for the calculation of the scavenging coefficient, a rainfall rate must be provided (in $\mathrm{mm\,h^{-1}}$). If the chosen type is `constant` or `none`, the rainfall rate or other information can be provided but will be ignored by the program. So, the meteorological data file finally looks like this:

```
[situation]

# Temperature (Celsius degrees)
```

```
Temperature = 10.

# Wind angle (degrees)
Wind_angle = 30.

# Wind speed (m/s)
Wind = 3.0

#Boundary height (m)
Boundary_height = 1000.

# Stability class
Stability = D

# Rainfall rate (mm/hr)
Rainfall_rate = 1.
```

In this example, there is only one meteorological situation described. Others can be added simply by adding similar sections `[situation]` at the end of the file.

2. Species data file: it contains several sections, but not all are needed for the preprocessing. The needed sections are:

   - `[species]` Contains the list of all species.
   - `[scavenging]` Contains the list of the species for which scavenging occurs. The scavenging coefficient of the others is set to 0.
   - `[deposition]` Contains the list of the species for which deposition occurs. The deposition velocity of the others is set to 0.
   - `[scavenging_constant]` This section is needed when the type of parameterization chosen for the scavenging is `constant`. It contains the name of a species followed by the value of its scavenging coefficient (in s^{-1}). Only one species per line must be provided. All species listed in the section `[scavenging]` must be present (the order is not important), the others will be ignored.
   - `[scavenging_belot]` This section is needed when the type of parameterization chosen for the scavenging is `belot`. It contains the name of a species followed by two values corresponding to the coefficients a and b respectively in the Belot parameterization. Only one species per line must be provided. All species listed in the section `[scavenging]` must be present, the others will be ignored.
   - `[deposition_constant]` This section is needed when the type of parameterization chosen for the deposition is `constant`. It contains the name of a species followed by the value of its deposition velocity (in $m\,s^{-1}$). Only one species per line must be provided. All species listed in the section `[deposition]` must be present, the others will be ignored.

   A species file might look like this:

```
[species]
```

```
Caesium       Iodine


[scavenging]

Iodine        Caesium

[deposition]

Caesium        Iodine


[scavenging_constant]

Caesium: 1.e-4
Iodine: 1.e-4


[scavenging_belot]

Caesium: 2.8e-05  0.51
Iodine: 7e-05  0.69


[deposition_constant]

Caesium: 0.05e-2
Iodine: 0.5e-2
```

In addition, if the chosen type is `file` for scavenging or deposition, the path to a file containing all values for each species and meteorological situation is given (entry `File`). The file containing the scavenging coefficients, named for example `scavenging.dat`, contains as mainy sections `[situation]` as the meteorological datafile. In each section, the species names are followed by the value of their scavenging coefficient for the given situation. The file for deposition velocities (`deposition.dat`) is in the same form. It looks like:

```
[situation]

Caesium: 0.0
Iodine: 0.0

[situation]

Caesium: 0.00006
Iodine: 0.0002
```

Output File The output data file contains as many sections as there are meteorological situations. Each section `[situation]` contains the temperature, wind angle, wind speed, boundary

height and stability class that are provided. In addition, it contains the list of all species followed by their scavenging coefficient, and the list of all species followed by their deposition velocity. It looks like this:

```
[situation]

# Temperature (Celsius degrees)
Temperature = 10

# Wind angle (degrees)
Wind_angle = 30.

# Wind speed (m/s)
Wind = 3.

# Boundary height (m)
Boundary_height = 1000

# Stability class
Stability = D

# Scavenging coefficient of the species (s^-1)
 Scavenging_coefficient =
 Caesium  6.36257e-05  Iodine  0.000212514

# Deposition velocity of the species (m/s)
 Deposition_velocity =
 Caesium  0.0005  Iodine  0.005
```

Program gaussian-deposition_aer

The program `gaussian-deposition_aer` works the same way as the program `gaussian-deposition`, except that there are some more information specific to the aerosol species. The input and output files are the same as described in the section about `gaussian-deposition`, so in this section we will only describe the data that are added to the files described previously. One input file is needed in addition to the meteorological data and species data files. It is the diameter file (reference: `diameter.dat`) which contains the diameters of the aerosol particles.

Configuration File In the configuration file, the following information are added:

	`[data]`
`Diameter`	Path to the data file that contains the particle diameters.
	`[scavenging]`
`Type_aer`	Parameterization to be used to calculate the scavenging coefficients for aerosol species.
`File_aer`	Path to the file that contains the scavenging coefficients for aerosol species. Read only if `Type_aer` is set to `file`.

`Value`	Values to be used for a Slinn parameterization (choose between `best_estimate` and `conservative`).
	`[deposition]`
`Type_aer`	Parameterization to be used to calculate the deposition velocities for aerosol species.
`File_aer`	Path to the file that contains the deposition velocities for aerosol species. Read only if `Type_aer` is set to `file`.
`Velocity_part`	If the deposition velocities are user-defined (type `constant` or type `file`), is it the total deposition velocity, or only the diffusive part (the gravitational settling velocity is then added)? Put `total` or `diffusive`.

The parameterization type for the scavenging coefficient of aerosol species can be chosen between:

- `none`: the scavenging coefficient is set to 0. for all aerosol species,
- `constant`: the scavenging coefficient is constant for one given diameter and entered in the species file,
- `file`: the scavenging coefficients are read in a file, for each species, diameter and meteorological situation,
- `slinn`: the scavenging coefficient is calculated with a Slinn parameterization. In that case, the only input data that are used are the rainfall rate and the particle diameters.

Concerning the deposition velocity, the type can be chosen between:

- `none`: the deposition velocity is set to 0. for all aerosol species,
- `constant`: the deposition velocity is constant for one given diameter and entered in the species file,
- `file`: the deposition velocities are read in a file, for each species, diameter and meteorological situation.

In case the entry `Velocity_part` is set to `diffusive`, the gravitational settling velocity is calculated for each particle, given the density and the diameter (provided in the species file) and the pressure and temperature (provided in the meteorological data file). It is then added to the constant values provided by the user.

Input Files

1. Meteorological data file: it is the same as the one for `gaussian-deposition`. If the parameterization type for the deposition velocity calculation is `constant` and the `Velocity_part` is `diffusive`, the pressure must be provided (in Pa).
2. Diameter file: it contains the list of particle diameters (in µm). The first number is the diameter of index 0, the second of index 1, and so on. This is an example of diameter file:

```
#Diameter (micrometer)
[diameter]
0.1
1.
```

The diameter of index 0 corresponds to the value 0.1 µm, the diameter of index 1 to the value 1. µm and so on. When referring to a given diameter in the other data files, one has to give the corresponding index. Note that there is only one diameter file for all aerosol species. Therefore all particulate species are assumed to have the same diameter distribution. The diameter file can also be the main configuration file. In that case, the section `[diameter]` is simply added to the main configuration file.

3. Species file: it is the same as described before, but the sections described for `gaussian-deposition` concern only gaseous species. All data concerning aerosol species are added in the following sections:

   - `[aerosol_species]` Contains the list of all aerosol species.
   - `[scavenging_constant_aer]` This section is needed when the type of parameterization chosen for the scavenging for aerosol species is `constant`. In that case, the scavenging coefficient is assumed to be constant for one particle diameter. So the section contains the index of one diameter followed by the corresponding value of the scavenging coefficient (in s^{-1}). Only one diameter per line must be provided.
   - `[deposition_constant_aer]`This section is needed when the type of parameterization chosen for the deposition of aerosol species is `constant`. It contains the index of a diameter followed by the value of its deposition velocity (in $m\,s^{-1}$). Only one diameter per line must be provided.
   - `[density_aer]` It contains the density of the aerosol species. That is, the name of each aerosol species followed by the corresponding density (in $kg\,m^{-3}$). Only one species per line must be provided. This section is needed in order to calculate the gravitational settling velocity of a particle. The calculated deposition velocity of one species of a given diameter is therefore a combination of the diffusive part given in the section `[deposition_constant_aer]` and the gravitational settling velocity calculated by the program.

Note that while some gaseous species might not be concerned by scavenging or deposition, the loss processes are assumed to occur for all aerosol species. Therefore, there is no need of a section containing the species for which scavenging or deposition occur in the case of aerosol species, as it is the case for gaseous species. Here is an example of species file containing the sections dedicated to aerosol species:

```
[aerosol_species]

aer1
aer2
aer3


[scavenging_constant_aer]

# Scavenging coefficient for aerosol species ( Unit: seconds^(-1) )
# Depends on the diameter (first value: diameter index in file diameter.dat).
# Only one diameter per line.

0: 1.e-4
```

```
1: 2.e-4


[deposition_constant_aer]

# Dry deposition velocity (diffusive part) of the species (Unit: m/s)
# Depends on the diameter
# Only one diameter per line.

0: 0.05e-2
1: 0.5e-2


[density_aer]

# Particle density (aerosol species) (kg/m^3)
# Only one species per line.

aer1: 1.88
aer2: 1.
aer3: 4.93
```

In case the `Type_aer` is set to `file`, an additional file is needed for deposition velocities or scavenging coefficients. It is in the same form as for the gaseous species, except that a value is required for each species and diameter. It may look like this:

```
[situation]

aer1_0: 0.03
aer1_1: 0.012
aer1_2: 0.01
aer2_0: 0.022
aer2_1: 0.011
aer2_2: 0.01

[situation]

aer1_0: 0.098
aer1_1: 0.071
aer1_2: 0.058
aer2_0: 0.091
aer2_1: 0.07
aer2_2: 0.058
```

Output File The output file is the same file as the one for `gaussian-deposition`, except that the scavenging coefficients and deposition velocities of aerosol species are also written. One coefficient corresponds to a given species of a given diameter. It is written as "species-name"_"diameter-index" followed by the value of the corresponding scavenging coefficient (or

deposition velocity). The following example corresponds to a case with two gaseous species named “gas1” and “gas2” and three aerosol species named “aer1”, “aer2” and “aer3”. The diameter file is the same as displayed before, that is, contains two diameters. The output file looks like this:

```
[situation]

# Temperature (Celsius degrees)
Temperature = 10

# Pressure (Pa)
Pressure = 101325

# Wind angle (degrees)
Wind_angle = 30

# Wind speed (m/s)
Wind = 3

# Boundary height (m)
Boundary_height = 1000

# Stability class
Stability = D

# Scavenging coefficient of the gaseous species (s^-1)
Scavenging_coefficient =
gas1  0.0001    gas2  0.0001

# Deposition velocity of the gaseous species (m/s)
 Deposition_velocity =
 gas1  0.0005     gas2  0.005

# Scavenging coefficient of the aerosol species (s^-1)
 Scavenging_coefficient_aer =
 aer1_0  5.95238e-05      aer1_1  5.95238e-05    aer2_0  5.95238e-05
 aer2_1  5.95238e-05      aer3_0  5.95238e-05    aer3_1  5.95238e-05

# Deposition velocity of the aerosol species (m/s)
 Deposition_velocity_aer =
 aer1_0  2.11708     aer1_1  6.69479      aer2_0  1.54404
 aer2_1  4.88268     aer3_0  3.42833      aer3_1  10.8413
```

The value following “aer1_0” corresponds to the calculated coefficient for the species “aer1” and the diameter of index 0, that is, in the case of our diameter file, the diameter equal to 0.1 µm. The value following “aer1_1” corresponds to the coefficient for the species “aer1” and the diameter of index 1, that is, equal to 1 µm, and so on.

Chapter 4

Drivers

4.1 BaseDriver

BaseDriver is configured with a file which contains the displaying options for the simulation.

[display]	
`Show_iterations`	If activated, each iteration is displayed on screen.
`Show_date`	If activated, the starting date of each iteration is displayed on screen in format `YYYY-MM-DD HH:II` (notations from Section D.7).

4.2 PlumeDriver

It is the driver dedicated to the Gaussian plume model. The associated configuration file is the same as the one for the BaseDriver, and it is usually part of the model configuration file described in Section 5.1. The associated input data file describes the meteorological data (reference: `gaussian-meteo.dat`) for gaseous species and `gaussian-meteo_aer.dat` for aerosol and/or gaseous species. The meteorological data file contains the meteorological data that are needed. It can be the output file of the preprocessing program `gaussian-deposition`.

The meteorological data file describes one or several meteorological situations. For each situation, the driver calls the model to calculate the concentrations, that is, the stationary solution for the given meteorological situation. It is associated with two models: the GaussianPlume model for gaseous species only (described in Section 5.1) and the GaussianPlume_aer model which is the same model for aerosol and/or gaseous species (see Section 5.2).

4.3 PuffDriver

It is the driver dedicated to the Gaussian puff model. The associated configuration file is the same as the one for the BaseDriver, and it is usually part of the model configuration file described in Section 5.3. The associated input data file describes the meteorological data. It is the same file as for the plume model.

For each meteorological situation, the driver calculates the concentrations that depend on time. That is, for a given situation, it makes a loop on time and calls the model at each time step to calculate the current concentrations. It is associated with two models: the GaussianPuff model for gaseous species only (described in Section 5.3) and the GaussianPuff_aer model which is the same model for aerosol and/or gaseous species (see Section 5.4).

4.4 PlumeMonteCarloDriver

The "plume Monte Carlo" driver is similar to the `PlumeDriver`, except that it performs several simulations, with perturbed input data and with different parameterizations, for every meteorological situation. The simulation outputs are saved with the unit saver of type `domain_ensemble_forecast` (see Section 4.9.2). The configuration file for this driver is the same as for `PlumeDriver` plus this additional content:

`[uncertainty]`	
`File_perturbation`	Path to the perturbations configuration (see below).
`Number_samples`	Number of "Monte Carlo" simulations for every meteorological situation.
`Random_seed`	Seed assignment for NewRan library, either a given seed number (in $]0,1[$), or the path to the directory that contains NewRan seed files, or `current_time` for a seed that depends on the current CPU time.

The perturbations and changes in the parameterizations are described by a perturbation configuration file (entry `File_perturbation` of the main configuration file). This file contains:

`[boolean_option]`	
Model Boolean options	The probability with which the option should be set to true.
`[string_option]`	
Model string options	A list of possible values (say `A` and `B`) for the string option together with their probabilities (possibly in brackets). For example: `A (0.3)  B (0.7)`
`[numerical_value]`	
Model numerical values	A PDF description (see below).
`[source_data]`	
Source data	A PDF description (see below).

The accepted PDFs (probability density functions) descriptions are:

1. `Uniform a b` which is for a uniform distribution with support $[m - \mathtt{a}, m + \mathtt{b}]$ where m is the model unperturbed value;
2. `Uniform_relative a b` which is for a uniform distribution with support $[m \times \mathtt{a}, m \times \mathtt{b}]$ where m is the model unperturbed value;
3. `Normal s` which is for a normal distribution with the model unperturbed value as mean and with $\mathtt{s}/2$ as standard deviation;
4. `Log-normal s` which is for a log-normal distribution with the model unperturbed value as median and with $\mathtt{s}/2$ as standard deviation of the logarithm of the variable.

With the Polyphemus `GaussianPlume` and `GaussianPlume_aer` models, the available options and data that may be listed in the perturbation file are:

1. `[boolean_option]`: `gillani`, `hpdm`, `rural`, `day`, `scavenging`, `dry_deposition`, `plume_rise`, `breakup`;

2. [string_option]: option parameterization_std with possible values Briggs, Doury and similarity_theory; option deposition_model with possible values Chamberlain and Overcamp;
3. [numerical_value]: temperature, wind_angle, wind, inversion_height, friction_velocity, convective_velocity, boundary_height, lmo, coriolis;
4. [source_data]: rate, velocity, temperature, z, y, x, diameter.

Note that the driver makes the model read all input data the model may require. Indeed, any parameterization or option of the model may be selected a priori (even if it does not eventually appear in the perturbations configuration), so the model must read all possible input data in order to run with any possible parameterization or option.

4.5 MonteCarloDriver

The Monte Carlo driver performs several simulations with perturbed input data. The input data are perturbed by the PerturbationManager (see Section 4.11). The simulation outputs are saved with the unit saver of type domain_ensemble_forecast (see Section 4.9.2). The configuration file for this driver should contain:

	[MonteCarlo]
Number_ensemble	The number of samples.
	[perturbation_management]
Configuration_file	Name of the file that contains perturbations configuration (see Section 4.11 about PerturbationManager).

4.6 PerturbationDriver

This driver can replace BaseDriver (see Section 4.1) and allows to perturb some input fields like the temperature or the deposition velocities. The perturbations are described by a perturbation configuration file (entry Perturbation_file in the section [data] of the main configuration file). This file should contain:

	[general]
Field_list	The fields relating to species which will be perturbed.

Then, the perturbations must be defined:

	[AdditionalField]
Field_name	Name of the field (usually independent of the species) which will be perturbed, followed of perturbation type (add or multiply) and the perturbation value.
	[Field]
Species	Species name followed of perturbation type (add or multiply) and the perturbation value.

The name of the section [Field] must appear in Field_list. For example:

```
[general]
Field_list: DepositionVelocity SurfaceEmission

[AdditionalField]
Temperature   add   2.0

[DepositionVelocity]
O3    multiply  1.2
NO2   multiply  1.2

[SurfaceEmission]
NO    multiply  1.5
ETH   multiply  1.5
```

4.7 Data Assimilation Drivers

4.7.1 AssimilationDriver

It is the base driver from which all data assimilation drivers are derived. Data assimilation is the concept and methods that estimate model state from diverse available sources, e.g. model simulations, observations and statistics information, aiming at and validated by a better prediction. Data assimilation methods can roughly be catalogued into variational and sequential ones. For the former the variational principle applies. The objective can be defined by the discrepancy between model simulation and a block of observations, usually combined with a priori background knowledge. This can be theorized and solved efficiently by optimal control theory (`FourDimVarDriver`). The sequential methods make use of observations instantaneously. This is a filtering process, and filter theory (linear or nonlinear) applies (`OptimalInterpolationDriver`, `EnKFDriver` and `RRSQRTDriver`).

A typical data assimilation system consists of three components: model (physics), data (observation), and assimilation algorithm. The data assimilation drivers organize model and data to perform assimilations.

The associated configuration file is an extension of that of the model configuration file exemplified in Section 5.8. For data part, it has an additional section `[observation_management]`:

[observation_management]	
`Configuration_file`	Path to the file containing the configuration of the observation management. In the distribution (directory `processing/assimilation/`), choose between observation.cfg (to use observations) and observation-sim.cfg (to use simulated observations).

The value of `Configuration_file` can be set to `observation.cfg` if you use `GroundObservationManager` (see Section 4.10.1) and to `observation-sim.cfg` if you use `SimObservationManager` (see Section 4.10.2).

The data assimilation experiments are controlled by the following options.

[domain]	
`Nt`	Number of time steps for the whole simulation (assimilation and prediction).

	[data_assimilation]
With_positivity_requirement	Is positivity of the assimilated species concentrations required?
Nt_assimilation	Number of time steps for the assimilation period.

Nt is supposed to be greater than or equal to Nt_assimilation. From time step #0 to time step #Nt_assimilation-1 , assimilation is performed; and from step #Nt_assimilation to step #Nt-1, prediction is performed.

In many cases such as data assimilation and ensemble prediction, perturbed model simulations are needed. Perturbations are managed by PerturbationManager (see Section 4.11) reading an additional section,

	[perturbation_management]
Configuration_file	Name of the file that contains perturbation configurations.

4.7.2 OptimalInterpolationDriver

It is the driver dedicated to data assimilation applications using optimal interpolation algorithm. The optimal interpolation algorithm estimates model state status by minimizing the error variance of the estimation (called *analysis* in data assimilation terminology). It searches for a linear combination between *background* state (model simulations) and the *background departures*. The background departures are defined as the discrepancies between observations and background state. It involves with observation managements (described in Section 4.10) and storage managements of forecast and analysis results (see for instance Section 4.9.4). The background error covariance matrix can be either diagonal or generated by Balgovind correlation functions (see Section 5.8).

4.7.3 EnKFDriver

It is the driver dedicated to data assimilation applications using ensemble Kalman filter algorithm. It consists of two steps: forecast and analysis. It differs from optimal interpolation in that the *background error covariance* is flow-dependent and approximated by an ensemble of perturbed model forecast. The algorithm parameters are set in section [EnKF].

	[EnKF]
Number_ensemble	The number of samples in the ensemble.
With_observation_perturbation	If observations are perturbed for consistent statistics for analyzed ensemble.
With_ensemble_prediction	If ensemble prediction is supported.

The generation of the ensemble is detailed in PerturbationManager configurations (see Section 4.11).

4.7.4 RRSQRTDriver

It is the driver dedicated to data assimilation applications using reduced rank square root Kalman filter algorithm (RRSQRT). It consists of two steps: forecast and analysis. The *background error covariance* is flow-dependent and approximated by an explicit low rank representation. The algorithm parameters are set in section [RRSQRT].

[RRSQRT]	
Number_analysis_mode	The expected rank number (column number) of the square root (mode matrix) of forecast error covariance matrix.
Number_model_mode	The number of the columns of the square root of model error covariance matrix to be added to the mode matrix.
Number_observation_mode	The number of the columns of the square root of observation error covariance matrix to be added to the analyzed mode matrix.
Propagation_option	The option for the forecast of the columns of mode matrix. Only finite difference is supported.
Finite_difference_perturbation	Perturbation coefficient for mode forecast using finite difference method; set to 1.

Model perturbations are employed to generate the columns of the square root of model error covariance matrix (see PerturbationManager configurations in Section 4.11).

4.7.5 FourDimVarDriver

It is the driver dedicated to data assimilation applications using four-dimensional variational assimilation algorithm (4D-Var). The assimilation period is from time step 0 to Nt_assimilation-1. The optimal model state at initial time step is obtained by minimizing an objective function which is the background departure plus discrepancy between model simulations and observations during the assimilation period. The model is supposed to be perfect, thus no model error terms are considered. The gradient of the objective function is calculated efficiently using adjoint model of the underlying model. The algorithm parameters are set in section [4DVar].

[4DVar]	
Display_precision	Display precision for optimization results.
Jb_file	Name of the file that saves background departure during optimization.
Jo_file	Name of the file that saves observation discrepancy during optimization.
Gradient_norm_file	Name of the file that saves gradient norms during optimization.
With_trajectory_management	If the trajectory of model integration is saved to disk for adjoint integration.
Trajectory_delta_t	Trajectory time step in seconds.
Trajectory_file	Name of the file that saves model trajectory.
With_background_item	If the background term is taken into account in the cost function.
Read_inverse_background_matrix	Should the inverse of background error covariance matrix be read from disk?
File_background_inverse_matrix	Name of the file that stores the inverse of the background error covariance matrix.
File_background_matrix	Name of the file that stores the background error covariance matrix.

The parameters for numerical optimization algorithm are set in section [optimizer]

[optimizer]	
Type	Type of optimization solver; only BFGS is supported.
Maximal_iteration	The number of the maximal iteration for numerical optimization.
Display_iterations	If the optimization results during iteration are displayed.

4.8 Drivers for the Verification of Adjoint Coding

The three drivers AdjointDriver, GradientDriver, Gradient4DVarDriver are dedicated to the verifications of adjoint model. The gradient of a given objective function calculated by adjoint model is compared with the gradient calculated by finite difference. The following ratio is checked

$$\rho = \frac{J(x + \alpha h) - J(x)}{\alpha \langle \nabla_x J, h \rangle}$$

where J is the objective function, x is the control variable, h is the perturbation direction, α is the perturbation coefficient, and $\nabla_x J$ is the gradient calculated by adjoint model, $< \ >$ denotes inner product. With $\alpha \to 0$, the ratio ρ is supposed to approach to 1 with high precision, then becomes unstable due to round-off errors. In practice, $\alpha = D^{-i}, i \in [m, m+1, \ldots, n]$, where D is the decreasing factor (typical values are 2 and 10), m is the integer for largest perturbation (typical value is 0), and n is the integer for smallest perturbation.

4.8.1 AdjointDriver

The objective function is chosen to be the model output of a given grid point in model domain with respect to initial model status. The corresponding gradient can therefore be interpreted as the sensitivity. This driver aims at the verification of adjoint code obtained by automatic differentiation of underlying model code using O∂yssée (version 1.7). The following options in section [adjoint] provide flexible control of the verification.

[adjoint]	
Point_species_name	Species name of the selected point in model domain for sensitivity calculation.
Point_nx	x-index of the selected point for sensitivity calculation.
Point_ny	y-index of the selected point for sensitivity calculation.
Point_nz	z-index of the selected point for sensitivity calculation.
Norm_perturbation_vector	Norm of the initial perturbation vector.
With_random_perturbation	With random directions for the perturbation?
Decreasing_root	Decreasing factor of the sequence of perturbation vectors (D).
Start_index	Index for the calculation of the first decreasing ratio (m).
End_index	Index for the calculation of the last decreasing ratio (n).
With_left_finite_difference_checking	Checking left-side finite difference results?

Display_sensitivity	Display sensitivity results for the decreasing perturbation sequences?

Option With_trajectory_management, Trajectory_delta_t, Trajectory_file are similar to those for 4DVar in Section 4.7.5.

4.8.2 GradientDriver

The objective function is chosen to be the norm of the difference between model simulations and synthetic observations. This driver aims at the verification of the backward integration algorithm of adjoint model for gradient calculations. Options Norm_perturbation_vector, With_random_perturbation, Decreasing_root, Start_index, End_index, and With_left_finite_difference_checking in section [adjoint] have the same meanings as those in Section 4.8.1. Option Display_cost indicates whether the values of the objective function are displayed when perturbation decreases accordingly.

4.8.3 Gradient4DVarDriver

The objective function is chosen to be the observation discrepancy in the 4DVar objective function. This driver aims at the verification of the adjoint code of observation operator. All the options for this driver are same as those in Section 4.8.2

4.9 Output Savers

4.9.1 BaseOutputSaver

The saver BaseOutputSaver is configured with a file that contains one or several sections [save]. Each section is associated with one element of a list of output-saver units managed by BaseOutputSaver.

According to the value of Type in every section, different saver units are called. Note however that a group attribute can be set in BaseOutputSaver (the default being all, and the other choices being forecast and analysis) and that only savers with the same group are called.

Some parameters must be provided for any kind of savers:

[save]	
Species	Chemical species to be saved. If it is set to all, concentrations for all species are saved.
Date_beg	The date from which the concentrations are saved. If concentrations are averaged, the first step at which concentrations are actually saved if not Date_beg, but Date_beg plus the number of steps over which concentrations are averaged. If the value - 1 is supplied, Date_beg is set at the start of the simulation.
Date_end	The last date at which concentrations may be saved. If the value - 1 is supplied, Date_end is set at the end of the simulation.
Interval_length	The number of steps between saves.
Type	The type of saver, see Table 4.16 for details.
Output_file	The full path of output files, in which &f will be replaced by the name of the chemical species. Note that the directory in which the files are written must exist before the simulation is started.

Note that Species, Date_beg, Date_end, Interval_length must appear before Type. After Type, put additional options relevant for the chosen output saver.

Here is a list of all types of saver units available at the moment:

Table 4.16: Types of saver

domain	To save entire vertical layers.
domain_aer	The same as domain but for aerosol species.
domain_assimilation	The same as domain but for data assimilation applications.
domain_prediction	The same as domain_assimilation but for model predictions based on model state at the end of the assimilation period.
domain_ensemble_forecast	The same as domain but for ensemble forecast applications.
domain_ensemble_analysis	The same as domain but for ensemble analysis applications.
nesting	To perform nested simulations.
nesting_aer	The same as nesting but for aerosol species.
subdomain	To save concentrations only for an horizontal subdomain.
subdomain_aer	The same as subdomain but for aerosol species.
indices_list	To save at a list of points given by their indices.
indices_list_aer	To save at a list of points given by their indices but for aerosol species.
coordinates_list	To save at a list of points given by their coordinates.
coordinates_list_aer	To save at a list of points given by their coordinates but for aerosol species.
wet_deposition	To save entire wet deposition fluxes.
dry_deposition	To save entire dry deposition fluxes.
wet_deposition_aer	The same as wet_deposition but for aerosol species.
dry_deposition_aer	The same as dry_deposition but for aerosol species.
backup	The backup gas species in order to restart.
backup_aer	The same as backup but for aerosol species.

4.9.2 SaverUnitDomain and SaverUnitDomain_aer

The output saver SaverUnitDomain defines an output-saver unit when Type is set to either "domain", or "domain_ensemble_forecast" and "domain_ensemble_analysis" in case of ensemble applications. This output saver requires additional parameters presented in the table below.

[save]	
Levels	A list of integers that determines the vertical layers to be saved. Note that 0 is the first layer. Remember that the heights you specified in the file levels.dat are those of the level interfaces, while concentration are saved in the middle of each levels.
Averaged	Should concentrations be averaged over Interval_length? If not, instantaneous concentrations are saved.
Initial_concentration	Should initial concentrations be saved? This option is only available if concentrations are not averaged.

For aerosol species, the saver should be SaverUnitDomain_aer and the Type "domain_aer". The section [save] is very similar to the one for gaseous species, except that you have to specify for which diameters the concentrations are saved. Hence, the list of species to be saved looks like this:

```
Species: aer1_{0} aer 1_{2} aer2_{0-1}
```

In that case, the species named "aer1" is to be saved for the diameter of indices 0 and 2, and "aer2" for the diameters of indices 0 and 1.

In `Output_file`, `&f` will be replaced by the species name and `&n` by the bin index. You can use any symbol which is not a delimiter (or even nothing) to separate the species name from the bin index, even though `&f_&n.bin` is the advised form.

If `Species` is set to "all" the concentrations will be saved for all aerosol species and for all diameters.

4.9.3 SaverUnitSubdomain and SaverUnitSubdomain_aer

These saver units allow the user to save concentrations only over an horizontal subdomain (for example, if they perform a simulation over the whole of Europe but only want the concentrations over one country or region). Their `Type` is "subdomain" and "subdomain_aer" respectively. The user must provide between which indices for x and y they want to save concentrations. The specific parameters for these saver units are:

	[save]
`Levels`	A list of integers that determines the vertical layers to be saved. Note that 0 is the first layer. Remember that the heights you specified in the file `levels.dat` are those of the level interfaces, while concentration are saved in the middle of each levels.
`Averaged`	Should concentrations be averaged over `Interval_length`? If not, instantaneous concentrations are saved.
`Initial_concentration`	Should initial concentrations be saved? This option is only available if concentrations are not averaged.
`i_min`	Minimum longitude index of the subdomain.
`i_max`	Maximum longitude index of the subdomain.
`j_min`	Minimum latitude index of the subdomain.
`j_max`	Maximum latitude index of the subdomain.

4.9.4 SaverUnitDomain_assimilation

The output saver `SaverUnitDomain_assimilation` defines an output-saver unit similar to `SaverUnitDomain`, except that it requires additional parameters presented in the table below.

	[save]
`Date_file`	The full path name of the file that stores the date sequences of the assimilation results.

The `group` attribute of the output saver `SaverUnitDomain_assimilation` is set to "analysis", whereas the `group` attributes of other saver units are set to "forecast" by default. Its `Type` is "domain_assimilation".

4.9.5 SaverUnitDomain_prediction

The output saver `SaverUnitDomain_prediction` defines an output-saver unit similar to `SaverUnitDomain`. The `group` attribute of the output saver `SaverUnitDomain_prediction` is set to "prediction". Its `Type` is "domain_prediction". It works in a similar way to

SaverUnitDomain_assimilation, except that it is designed for the storage of model predictions starting from analyzed model state at the end of the assimilation period.

4.9.6 SaverUnitNesting and SaverUnitNesting_aer

The saver units SaverUnitNesting and SaverUnitNesting_aer are used to perform nested simulations. That means that the results of a first simulation on a large domain are interpolated and saved at the boundary of a subdomain and are then used as boundary conditions for a second simulation on the subdomain.

The two simulations are quite "normal" except that:

- **For the first one:** additional concentrations have to be saved using SaverUnitNesting and/or SaverUnitNesting_aer.
- **For the second one:** boundary conditions from the first simulation have to be provided the usual way in polair3d-data.cfg.

Refer to the files in processing/nesting for concrete examples. Files ending with "-nesting" are for the first simulation and files ending with "-nested" are for the second one.

If the saver unit is of Type "nesting" or "nesting_aer", the additional parameters needed in the section [save] are presented in the table below.

[save]	
x_min	Origin of the subdomain along x.
Delta_x	Step along x for the subdomain.
Nx	Number of points along x for the subdomain.
y_min	Origin of the subdomain along y.
Delta_y	Step along y for the subdomain.
Ny	Number of points along y for the subdomain.
levels	File giving the interfaces of the layers for the subdomain.
Nz	Number of layers in the subdomain.

In Output_file &f and &n are replaced as for SaverUnitDomain or SaverUnitDomain_aer and &c is replaced by the direction along which the boundary conditions were interpolated (that means that &c is replaced by x, y or z).

4.9.7 SaverUnitPoint and SaverUnitPoint_aer

The saver units SaverUnitPoint and SaverUnitPoint_aer are used to save concentrations at a list of given points. There are two possible types: indices_list saves concentrations at given indices in the simulation grid (provided in the main configuration file in section [domain]) and coordinates_list saves concentrations at given (Cartesian) coordinates. In case the saver is of type coordinates_list the simulation grid is still read but is not used, as concentrations are computed directly at each point of interest.

The list of points where concentrations are to be saved has to be specified in the section [save] of the saver configuration file. It has to be written just after the line containing the field Output_file. The list begins with the field Indices in case the saver type is indices_list and Coordinates if the type is coordinates_list. In both cases, the list must end with a line containing the field Point_file, which is used to specify a file name where the list of all points where concentrations are saved is to be written during the simulation. This looks like:

```
Type: indices_list
Levels: 0 2

Output_file: <Results>/&f.bin

Indices:

0   0
5   15
20  25
30  30

Point_file: <Results>/point.txt
```

One line corresponds to one point. There can be either two or three indices. In case there are two indices on the line, the first one corresponds to index along y, the second one to the index along x, and concentrations are saved at this point for each vertical level specified in the field `Levels`. In the previous example, the file `<Results>/point.txt` will be created during the simulation and look like:

```
# z   y    x

  0   0    0
  2   0    0
  0   5    15
  2   5    15
  0   20   25
  2   20   25
  0   30   30
  2   30   30
```

In case there are three indices on a line, concentrations are saved only at the specified point, no matter what the field `Levels` contains. The first value on the line corresponds to index along z, the second to index along y and the third to index along x. Note that lines containing two or three values can be entered in any order. The output binary file containing concentrations will simply follow the order given in the file `<Results>/point.txt` for each time step. It means that the resulting binary file will be of size $N_t \times N_{point}$ where N_t is the number of time steps to be saved, and N_{point} is the number of points where concentrations are saved.

To save coordinates instead of indices, one simply has to change the type to `coordinates_list`, to add the field `Levels_coordinates` to specified values of z where concentrations are to be saved, and to write the field `Coordinates` instead of `Indices`. The field `Levels` is still read but not used. Here is an example:

```
Levels: 0
Levels_coordinates: 1.5

Output_file: <Results>/&f.bin

Coordinates:
```

```
470.0  535.0
470.7  535.0
470.5  535.1
1.5    470.8   535.4
2.5    470.8   535.4
4.5    470.8   535.4
7.5    470.8   535.4
10.5   470.8   535.4
13.5   470.8   535.4
17.5   470.8   535.4

Point_file: <Results>/point.txt
```

Note that coordinates are entered in meters, first z, then y, then x, or just y then x. In the previous example, every point for which only two coordinates are entered is saved at 1.5 meters above ground. For one point, one wished to save concentrations at different heights above ground, so heights have been explicitly written.

When dealing with aerosol species, one just has to put `indices_list_aer` or `coordinates_list_aer` instead of `indices_list` or `coordinates_list` respectively.

4.9.8 SaverUnitWetDeposition and SaverUnitDryDeposition

The output savers `SaverUnitWetDeposition` and `SaverUnitDryDeposition` define output-saver units when `Type` is set to "wet_deposition" or "dry_deposition" and both require additional parameters presented in the table below.

	`[save]`
`Averaged`	Should concentrations be averaged over `Interval_length`? If not, instantaneous concentrations are saved.
`Initial_concentration`	Should initial concentrations be saved? This option is only available if concentrations are not averaged.

If `Species` is set to "all" the deposition fluxes will be saved for all scavenged or dry deposited species.

4.9.9 SaverUnitWetDeposition_aer and SaverUnitDryDeposition_aer

The output savers `SaverUnitWetDeposition_aer` and `SaverUnitDryDeposition_aer` define output-saver units when `Type` is set to "wet_deposition_aer" or "dry_deposition_aer". The section `[save]` is very similar to the one for gaseous species, except that you have to specify for which diameters the deposition fluxes are saved. Hence, the list of species to be saved looks like this:

```
Species: aer1_{0} aer 1_{2} aer2_{0-1}
```

In that case, the species named "aer1" is to be saved for the diameter of indices 0 and 2, and "aer2" for the diameters of indices 0 and 1.

In `Output_file`, `&f` will be replaced by the species name and `&n` by the bin index. You can use any symbol which is not a delimiter (or even nothing) to separate the species name from the bin index, even though `&f_&n.bin` is the advised form.

If `Species` is set to "all" the deposition fluxes will be saved for all aerosol species and for all scavenged or dry deposited diameters.

4.9.10 SaverUnitBackup and SaverUnitBackup_aer

The output savers `SaverUnitBackup` and `SaverUnitBackup_aer` are respectively similar to `SaverUnitDomain` and `SaverUnitDomain_aer` output-savers: they save gas and aerosol concentrations over the entire domain. The difference is that latter output-savers are intended for post-treatment whereas formers for eventually be able to restart a simulation as if it had not stopped.

Therefore, all gas and aerosol concentrations are saved and not averaged, only one simulation time step is saved and each backup overwrites the latter one. A "date" file stores the current date and iteration of backup files.

All backup files are buffered in files with extension ".buf". These buffers are only intended for the case the simulation breaks during the backup saving. In this case the "date" file will contain the message "!! BACKUP SAVING NOT FINISHED !!" which means that buffer files have to be used instead of backup ones. These buffer files are only needed at run time and are removed at the end of simulation.

The output savers `SaverUnitBackup` and `SaverUnitBackup_aer` are configured with a `[save]` section:

```
[save]

Type: backup
Interval_length: 10
Output_file: backup/&f.bin
Date_file: backup/date_backup

[save]

Type: backup_aer
Interval_length: 10
Output_file: backup/&f_&n.bin
Date_file: backup/date_backup_aer
```

The backup output-savers are selected by setting `type` to `backup` or `backup_aer`. The number of time steps between two backups is set in `Interval_length`. In order to be able to correctly restart a simulation this number has to be large enough compared to that of other output-savers. Furthermore the backup time must overlap the save time of other output-savers. For example if a simulation makes averaged savings every six time steps, the backup `Interval_length` has to be a multiple of six and at least six.

How to restart? Open the "date" file, pick up the backup date and replace the beginning date of `polair3d.cfg` main configuration file with it. Then go into `polair3d-data.cfg` and change the `initial_condition` and `initial_condition_aer` sections so that they points to the backup files. Modify the `[save]` sections in order to keep previous output files.

Remark A restart must give exactly the same results as if the simulation had not stopped. Nevertheless vector concentrations are stored in memory in double precision whereas backups are

written on disk in simple precision so that on restart you cannot avoid roundoff errors between simple and double precision.

4.10 Observation Managers

The observation managers deal with available observational data at different locations and dates. These managers are designed to prepare for applications related to observation treatments, especially for data assimilation. The observation operator are implemented for the mapping from observation space into model space. For a given date, these managers retrieve observation data values and the corresponding statistical information, e.g. observational error covariances.

4.10.1 GroundObservationManager

The `GroundObservationManager` is dedicated to ground observation managements.

	`[general]`
`Species`	Name of observed species. The current version deals with only one observed species.
`Error_variance`	Error variance for the observed species.
`With_spatial_interpolation`	Should observations be interpolated at adjacent model grid points?
`With_perturbation`	Should the observation be perturbed?
`Perturbation_scale`	If `With_perturbation` is set to "yes", gives the amplitude of the perturbation.
	`[stations]`
`Nstations`	Total number of stations.
`Stations_file`	File containing station information (code, name, latitude, longitude and altitude). `&s` in path names is replaced by species name specified in `[general]` section.
`Input_directory`	Directory where the observations are stored.

4.10.2 SimObservationManager

The `SimObservationManager` is dedicated to synthetic observation managements. Library NewRan is needed for random number generations. Note that NewRan is not included in the distribution, and it is the user's duty to install NewRan. The associated configuration file is an extension of that of `GroundObservationManager`. The additional sections are mainly for data specifications of the binary data files.

	`[simulation_manager]`
`Simulation_option`	Specifies how observations are provided. The current version deals only with observations at ground stations.
`Input_file`	Files containing the observation data. They usually are generated by certain reference run of Polair3D. `&s` in path names is replaced by species name.
`Date_min`	Starting date for the simulation results in data files.
`Delta_t`	Time step in seconds for the simulation results in data files.
`Levels`	Levels for the simulated data in files.

`Initial_concentration`	Flag that indicates whether initial concentrations are included in data file.

4.11 Perturbation Manager

The `PerturbationManager` is dedicated to perturbation managements. It reads the perturbation configurations, and performs perturbations. The concerning fields are then updated according to perturbation results for new model simulations in diverse applications such as data assimilation and ensemble predictions. Library NewRan is needed for random number generations.

The perturbation fields are defined in configuration file of which the name can be read from section `[perturbation_management]` in the driver configuration file.

`[general]`	
`Fields`	Perturbation field list that depends on species, e.g. DepositionVelocity, PhotolysisRate, SurfaceEmission, and BoundaryCondition.
`Rand_seed`	Seed assignment for NewRan library, either a given seed number (in $]0, 1[$), or the path to the directory that contains NewRan seed files, or `current_time` for a seed that depends on the current CPU time.
`Field_maximum_spread`	Every random number for field perturbations cannot exceed the mean plus or minus 'Field_maximum_spread' times the standard deviation.
`Observation_maximum_spread`	Every random number for observation perturbations cannot exceed the mean plus or minus 'Observation_maximum_spread' times the standard deviation.

For fields listed in section `[AdditionalField]` e.g. `Attenuation` and `VerticalDiffusionCoefficient`, the perturbations do not depend on species. For each additional field, or each species of species-dependent fields, the perturbation is performed according to lognormal (LN) law with given standard deviation. For normal law (N), the relative standard deviation is given, e.g 0.01 for `Temperature`, except for the field `WindAngle` (in degrees). The probability distribution types are listed in the `PDF` column, and the standard deviations are listed in the `Parameter` column. Sometimes there are two additional columns for species-dependent fields indicating correlated species, and correlation coefficient. The correlation coefficients can only be set to 1.

Chapter 5

Models

There are four major types of models: Gaussian models (see Section 5.1, 5.2, 5.3 and 5.4), Polair3D models (see Section 5.5, 5.6, and 5.7), Castor models (see Section 5.9) and Lagrangian models (see Section 5.13). All variants of a model have the same principles but can deal with various applications and phenomena.

Polair3D models were the first implemented in Polyphemus. They allow, as well as Castor models, to compute the advection and diffusion of pollutants at a large scale and can integrate various additional phenomena (such as photochemical chemistry or deposition). Gaussian models have been added to perform simulation at a local scale of the effect of a continuous (plume) or instantaneous (puff) source of pollutant.

Lagrangian stochastic models were the last implemented in Polyphemus. They allow to compute the passive dispersion of pollutants. Scavenging or deposition cannot be taken into account yet.

As for now, Castor models only deal with gaseous species, while the other models deal with gaseous or aerosol species.

5.1 GaussianPlume

Model `GaussianPlume` is the Gaussian plume model for gaseous species only. The associated program to be run is `plume` and it is configured with one configuration file (`plume.cfg`) and four data files (`plume-source.dat`, `plume-level.dat`, `gaussian-meteo.dat` and `gaussian-species.dat`). The configuration file provides the paths to the four other files. Basically, given a series of continuous point sources, it calculates the concentration of each species along a specified grid. There are several output files, one for each species, that are binary files. The way results are saved is described in an additional configuration file which corresponds to the file described in Section 4.9 (reference: `plume-saver.cfg`).

In these configuration files, there are entries that are not relevant for the Gaussian model but that must be provided anyway. In descriptions of configuration files (below), they are described as `irrelevant`.

5.1.1 Configuration File: `plume.cfg`

	`[domain]`
`Date_min`	**Irrelevant.** Provide a date.
`Delta_t`	**Irrelevant.** Provide any number.
`Nt`	**Irrelevant.** Provide an integer.

`x_min`	Abscissa in meter of the center of the lower-left cell.
`Delta_x`	Step length along x (in m).
`Nx`	Number of cells along x (integer).
`y_min`	Ordinate in meter of the center of the lower-left cell.
`Delta_y`	Step length along y (in m).
`Ny`	Number of cells along y (integer).
`Nz`	Number of vertical levels (integer).
`Vertical_levels`	Path to the file that defines vertical levels heights.
`Land_category`	Land category (choose between `rural` and `urban`). Relevant only when standard deviations are computed with Briggs parameterization.
`Time`	Choose whether it is nighttime (`night`) or daytime (`day`).
`Species`	Path to the file that defines involved species.
	`[gaussian]`
`With_plume_rise`	Is plume rise taken into account?
`With_plume_rise_breakup`	Is unstable and neutral breakup taken into account when computing plume rise?
`With_radioactive_decay`	Is radioactive decay taken into account?
`With_biological_decay`	Is biological decay taken into account?
`With_scavenging`	Is scavenging taken into account?
`With_dry_deposition`	Is dry deposition taken into account?
`Sigma_parameterization`	Parameterization used to compute standard deviations (`Briggs` for Briggs parameterization, `Doury` for Doury parameterization, and `similarity_theory` for a parameterization based on similarity theory).
`Above_BL`	Is a special formula used for the standard deviation above the boundary layer? Currently, only "Gillani" can be entered to provide a special formula. Otherwise, provide "none".
`With_HPDM`	Only relevant when similarity theory parameterization is used. It uses alternative formulae from the HPDM model to compute the standard deviations. It is recommended in the case of elevated sources (about 200 meters).
`Plume_rise_parameterization`	Parameterization used to compute the plume rise: `HPDM`, `Holland` or `Concawe`.
`File_meteo`	Path to the file containing the meteorological data.
`File_source`	Path to the file that describes the sources.
`File_correction`	File containing the correction coefficients (used with line sources and gaseous species only).
	`[deposition]`
`Deposition_model`	Model used to take dry deposition into account (`Chamberlain` for Chamberlain model, `Overcamp` for Overcamp model).
`Nchamberlain`	Number of points to calculate the Chamberlain integral (integer). Relevant only when dry deposition with Chamberlain model is taken into account.
	`[output]`

`Configuration_file`	Path to the configuration for the output saver.

Note: The Chamberlain integral for the calculation of dry deposition is discretized and approximated as a sum. The integer that is provided corresponds to the number of terms of the sum in the plume model. In the puff model, it is incremented at each time step, so as to have a number of points consistent with the range of the integral (that is, not to have too many points to discretize an integral whose range is very small).

5.1.2 Source Description: `plume-source.dat`

The point emission file used by the Gaussian plume model are described in Section 5.15. There are as many sections as sources, and they can be only of type "continuous". One source can emit several species. The beginning and ending date are read, but not used, since this is a stationary model. The source file can contain a list of point sources provided by the user or a discretized line source. In that case, it corresponds to the output file of the discretization preprocessing program `discretization`.

5.1.3 Vertical Levels: `plume-level.dat`

Vertical levels are defined in a single data file. They are defined by their interfaces. This means that the file contains `Nz+1` heights, where `Nz` is the number of levels specified in the main configuration file. The concentrations are computed at layers mid-points.

5.1.4 Species: `gaussian-species.dat`

Species are listed in the section `[species]` of a data file (the same as the species data file used in the preprocessing program `gaussian-deposition`, see Section 3.9.2). When radioactive or biological decay is taken into account, a section containing the half-life times of the species has to be provided. The section `[half_life]` contains the list of all species followed by their half-life time in days for radioactive decay (put 0. in the case of non-radioactive species). Provide only one species per line. The section `[half_life_time]` contains two values following each species name, the first corresponding to its biological half-life time (in s) during daytime and the second to the value during nighttime (put 0. in the case of non biological species). Here is an example:

```
[species]

# List of the species
Caesium     Iodine    bio1


[half_life]

# Half-life of the species (Unit: days)
# 0 coresponds to non-radioactive species
Caesium: 1.1e4
Iodine: 8.04
bio1: 0.

[half_life_time]
```

```
# Half-life of the species ( Unit: seconds )
# First value: day, second value: night
# 0 corresponds to non-radioactive species.
Caesium: 0. 0.
Iodine:  0. 0.
bio1: 1000 500
```

In that case we have two radioactive species, "Caesium" and "Iodine", and one biological species, "bio1".

If scavenging is taken into account, sections `[scavenging]` and `[scavenging_constant]` must be added. The section `[scavenging]` contains the name of all species for which scavenging occur and `[scavenging_constant]` their constants in s^{-1}.

5.1.5 Meteorological data file: `gaussian-meteo.dat`

This file contains basic meteorological information needed to run Gaussian models. In case there are scavenging and deposition, it is the output file of preprocessing program `gaussian-deposition` described in Section 3.9.2. Information that are always needed are:

- Temperature (°C)
- Wind angle (° from x axis)
- Wind speed ($m\,s^{-1}$)
- Boundary layer height (m)

These information are provided inside a section `[situation]`, and the meteorological data file contains as many sections are there are situations. The boundary height is always needed: during daytime, reflections on the inversion layer are performed. It is also used to computed the standard deviations in the case of the parameterization based on similarity theory.

```
[situation]

# Temperature (Celsius degrees)
Temperature = 15

# Wind angle (degrees)
Wind_angle = -100

# Wind speed (m/s)
Wind = 0.5

# Boundary layer height (m)
Boundary_height = 500.0
```

In case the Briggs parameterization for standard deviation is used, a stability class between 'A' (very unstable) and 'F' (very stable) has to be provided.

```
# Stability class
Stability = A
```

In case the standard deviations are computed with similarity theory, more information have to be provided:

- Friction velocity ($\mathrm{m\,s^{-1}}$)
- Convective velocity ($\mathrm{m\,s^{-1}}$)
- Monin-Obukhov length (m)
- Coriolis parameter ($\mathrm{s^{-1}}$)

If not known, the Coriolis parameter can be set to $10^{-4}\mathrm{s^{-1}}$. Note that in the case there is plume rise and `With_plume_rise_breakup` is set to "yes", the convective velocity and friction velocity are also needed for plume rise computation with HPDM or Concawe formulae (whatever the standard deviation parameterization is).

```
[situation]

# Friction velocity (m/s)
Friction_velocity = 0.37

# Convective velocity (m/s)
Convective_velocity = -0.81

# Monin Obukhov Length (m)
LMO = 120.0

# Coriolis parameter (/s)
Coriolis = 1.4e-04
```

5.1.6 Correction coefficients file: `correction_coefficients.dat`

The correction coefficient file is used for line sources and gaseous species only. It contains coefficients for each stability classes and land categories of Briggs parameterization. These coefficients are used to reduce the error induced by the Gaussian line source analytical formula.

5.2 GaussianPlume_aer

It is the Gaussian plume model for aerosol species. The corresponding program is `plume_aer`. It can be run when there are aerosol species only, or both aerosol and gaseous species. It takes the same input files as the Gaussian plume model, except that they contain in addition some sections dedicated to aerosol species. It takes in addition another input file that describes the diameters of particles (file `diameter.dat` already described in Section 3.9.2). The output files are binary files, one for each gaseous species and one for each couple (aerosol species, diameter). The way results are saved is described in an additional configuration file (reference: `plume-saver_aer.cfg`) described in Section 4.9.

5.2.1 Configuration File: `plume_aer.cfg`

It is exactly the same file as the configuration file described in Section 5.1. The only data that may differ are the paths to the input files.

5.2.2 Source Description: `plume-source_aer.dat`

It is the same file as the source file for gaseous species, except that obviously some (or all) emitted species will be particulate species. The corresponding sections are named `[aerosol_source]`. However, some lines are different for aerosols:

- The species is given after the key word "Species_name" instead of "Species".
- Only one species per source can be given.

5.2.3 Vertical Levels: `plume-level.dat`

It is the same file as in Section 5.1.

5.2.4 Species: `gaussian-species_aer.dat`

The section `[species]` lists the gaseous species, and the section `[aerosol_species]` lists the aerosol species. In the case of radioactive or biological decay, the sections are the same as described in Section 5.1 and contain the half-life times of both gaseous and aerosol species.

5.2.5 Diameters: `diameter.dat`

See Section 3.9.2.

5.2.6 Meteorological data: `gaussian-meteo.dat`

See Section 5.1.5.

5.3 GaussianPuff

Model `GaussianPuff` is the Gaussian puff model for gaseous species only. The associated program to be run is `puff` and it is configured with one configuration file (`puff.cfg`) and four data files (`puff.dat`, `puff-level.dat`, `gaussian-meteo.dat` and `gaussian-species.dat`). The configuration file provides the paths to the four other files. Basically, given a series of instantaneous puffs emitted at different times, it calculates the concentration of each species along a specified grid. There are several output files, one for each species, that are binary files. (same as in the Gaussian plume model, and fully described in Section 4.9).

5.3.1 Configuration File: `puff.cfg`

	`[display]`
`Show_date`	**Irrelevant.** Provide any Boolean.
	`[domain]`
`Date_min`	Provide the simulation starting date (see Section D.7).
`Delta_t`	Time step of the simulation (in seconds).
`Nt`	Number of time steps (integer).
`x_min`	Abscissa in meter of the center of the lower-left cell.
`Delta_x`	Step length along x (in m).
`Nx`	Number of cells along x (integer).
`y_min`	Ordinate in meter of the center of the lower-left cell.

`Delta_y`	Step length along y (in m).
`Ny`	Number of cells along y (integer).
`Nz`	Number of vertical levels (integer).
`Vertical_levels`	Path to the file that defines vertical levels heights.
`Land_category`	Land category (choose between `rural` and `urban`).
`Time`	Choose whether it is nighttime (`night`) or daytime (`day`).
`Species`	Path to the file that defines involved species.
	`[gaussian]`
`With_plume_rise`	Is plume rise taken into account?
`With_plume_rise_breakup`	Is unstable and neutral breakup taken into account when computing plume rise? (read only if `With_plume_rise` is set to 'yes').
`With_radioactive_decay`	Is radioactive decay taken into account?
`With_biological_decay`	Is biological decay taken into account?
`With_scavenging`	Is scavenging taken into account?
`With_dry_deposition`	Is dry deposition taken into account?
`With_increasing_sigma`	Option to force standard deviations to increase in time in case meteorological data are not stationary. This is useful when coupling the puff model with the plume-in-grid model.
`With_chemistry`	Is there chemistry within the puffs?
`With_puff_interaction`	Is there chemical interaction between two overlapping puffs? (read only if `With_chemistry` is set to 'yes').
`Sigma_parameterization`	Parameterization used to compute standard deviations (`Briggs` for Briggs parameterization, `Doury` for Doury parameterization, and `similarity_theory` for a parameterization based on similarity theory).
`Above_BL`	Is a special formula used for the standard deviation above the boundary layer? Currently, only "Gillani" can be entered to provide a special formula. Otherwise, provide "none".
`With_HPDM`	Only relevant when similarity theory parameterization is used. It uses alternative formulae from the HPDM model to compute the standard deviations. It is recommended in the case of elevated sources.
`Plume_rise_parameterization`	Parameterization used to compute the plume rise: `HPDM`, `Holland` or `Concawe` (read only if `With_plume_rise` is set to 'yes').
`File_meteo`	Path to the file containing the meteorological data.
`File_puff`	Path to the file that contains the puff data.
`Delta_t_puff`	Time step between two puff emissions (can be greater than, or equal to, the simulation time step).
	`[deposition]`
`Deposition_model`	Model used to take dry deposition into account (`Chamberlain` for Chamberlain model, `Overcamp` for Overcamp model)
`Nchamberlain`	Number of points to calculate the Chamberlain integral (integer). Relevant only when dry deposition with Chamberlain model is taken into account.
	`[output]`
`Configuration_file`	Path to the configuration for the output saver.

`With_output_plume_mass`	If 'yes', the total plume mass is saved into a binary file of size $N_t \times N_{species}$ (only with chemistry).
`File_mass`	Path to the binary file to save the total plume mass (read only if `With_output_plume_mass` is set to 'yes').

5.3.2 Puff Description: `puff.dat`

The point emission file used by the Gaussian puff model are described in Section 5.15. There are as many sections as sources, and they can be of type "puff" or "continuous". In the later case, the continuous source is discretized into a series of puffs with the time step given in the main configuration file with field `Delta_t_puff`. If the simulation time step is Δt, the effective time step between two puffs is $Nt_{\text{puff}} \times \Delta t$, where $Nt_{\text{puff}} = \max(\Delta t_{\text{puff}}/\Delta t, 1)$. Hence, it can only be equal to or greater than the simulation time step.

Every puff time step, if the source is still emitting at that time, and if the source rate is R (in mass unit per second), a new puff of quantity $Q = R \times Nt_{\text{puff}} \times \Delta t$ is emitted. One source can emit several species.

Note that the puff file can contain a list of puffs corresponding to a discretized line source or trajectory. In that case, it corresponds to the output file of the discretization preprocessing program `discretization`.

5.3.3 Vertical Levels, Species and Meteorological data

They are exactly the same files as those described in Section 5.1. When using chemistry, some additional information are needed:

- In the species configuration file, a section `[photolysis]` contains the list of species with photolysis.
- In the configuration file for meteorological data, the pressure (in Pa), attenuation coefficient, and specific humidity in $\text{kg}\,\text{kg}^{-1}$ are needed.
- In the configuration file for meteorological data, after the species data for scavenging and deposition, the list of photolyis rates and background concentrations are needed. The list of photolysis rates must be provided for all species contained in the section `[photolysis]` of the species file (23 species for RACM mechanism), followed by the rate. The background concentrations can be provided only for species for which it is not equal to zero. It is the concentration of the species in the atmosphere, outside the puff. It is supposed to be homogeneous. During the simulation, the background species concentrations change with chemistry, and the species interact with the puff species.

Here is an example of the additional entries in the meteorological data file, where only the ozone background concentration is different from zero.

```
# Pressure (Pa)
Pressure = 101325.

# Attenuation
Attenuation = 0.99

# Specific humidity
```

```
Specific_humidity = 0.011

# Photolysis rates of the species
Photolysis_rate =
ALD 2.74581e-06 GLYform 4.58622e-05 GLYmol 4.63453e-05
H2O2 5.04327e-06 HCHOmol 2.95972e-05 HCHOrad 1.90028e-05
HKET 4.91193e-07 HNO3 2.13667e-07 HNO4 3.87569e-06 HONO 0.00131391
HOP 4.96263e-06 KETONE 4.91193e-07 MACR 0.000358261
MGLY 0.00013251 MHP 4.96263e-06 NO2 0.00731999 NO3NO 0.0193996
 NO3NO2 0.157498 O3O1D 1.06414e-05 O3O3P 0.0004013 ORGNIT 6.41837e-07
PAA 1.41212e-06 UDC 0.000397305

# Background concentrations of the species
Background_concentration =
O3 40.
```

5.4 GaussianPuff_aer

It is the Gaussian puff model for aerosol species. It can be run when there are aerosol species only, or both aerosol and gaseous species. It takes the same input files as the Gaussian puff model, except that they contain in addition some sections dedicated to aerosol species. It takes in addition another input file that describes the diameters of particles (file `diameter.dat` already described in the Section 3.9.2). The output files are binary files, one for each gaseous species and one for each couple (species, diameter).

5.4.1 Configuration File: `puff_aer.cfg`

It is exactly the same file as the configuration file described in Section 5.3. The only data that may differ are the paths to the input files.

5.4.2 Source Description: `puff_aer.dat`

It is the same file as the puff file for gaseous species described in Section 5.3, except that obviously some (or all) emitted species will be particulate species. The corresponding sections are named `[aerosol_source]`. However, some lines are different for aerosols:

- The species is given after the key word "Species_name" instead of "Species", and only one species per source can be given.
- The dates are not read, only the time (in seconds) after the beginning of the simulation when the puff is released, after the word "Release_time".
- Only puff sources can be treated, not continuous sources.

5.4.3 Vertical Levels, Species, Meteo and Diameters

Vertical level file and gaussian meteo file have been described in Section 5.1 and diameter files is the same as in Section 3.9.2. Species file is the same file as described for the plume model for aerosol species (Section 5.2.4).

5.5 Polair3DTransport

The model Polair3DTransport is configured with three configuration files (`polair3d.cfg`, `polair3d-data.cfg` and `polair3d-saver.cfg`) and two data files (`levels.dat` and `species.dat`). The main configuration file (`polair3d.cfg`) provides the paths to the four other files.

5.5.1 Main Configuration File: `polair3d.cfg`

The configuration file `polair3d.cfg` gives information on the domain definition and the options of the simulation:

	`[domain]`
`Date_min`	Starting date in any legal format (see Section D.7). The date can therefore include seconds.
`Delta_t`	Time step in seconds.
`Nt`	Number of iterations of the simulation (integer).
`x_min`	Abscissa of the center of the lower-left cell. Provide a longitude (in degrees) or, in case Cartesian coordinates are chosen, an abscissa in meters.
`Delta_x`	Step length along x, in degrees (longitude) or in meters (for Cartesian coordinates).
`Nx`	Number of cells along x (integer).
`y_min`	Ordinate of the center of the lower-left cell. Provide a latitude (in degrees) or, in case Cartesian coordinates are chosen, an ordinate in meters.
`Delta_y`	Step length along y, in degrees (latitude) or in meters (for Cartesian coordinates).
`Ny`	Number of cells along y (integer).
`Nz`	Number of vertical levels (integer).
`Vertical_levels`	Path to the file that defines vertical levels interfaces.
`Cartesian`	If activated, coordinates are Cartesian and in meters. Otherwise, coordinates are latitudes and longitudes in degrees.
`Species`	Path to the file that defines involved species and their chemical properties.
	`[options]`
`With_advection`	Are species advected?
`With_diffusion`	Are species diffused?
`With_air_density`	If activated, vertical wind is diagnosed from $\mathrm{div}(\rho V) = 0$ where ρ is the air density and V the wind, and the diffusion term is $\mathrm{div}\left(\rho K \nabla \frac{c}{\rho}\right)$ where c is the concentration and K is the diffusion matrix. If this option is not activated, it is assumed that ρ is constant and therefore disappears from the previous equations.
`With_initial_condition`	Are initial conditions provided for given species? If not, initial concentrations are set to zero.
`With_boundary_condition`	Are boundary conditions available for given species?
`With_deposition`	Is dry deposition taken into account?

`With_point_emission`	Are point emissions provided?
`With_surface_emission`	Are emissions at ground provided?
`With_additional_surface_emission`	Are additional emissions at ground provided?
`With_volume_emission`	Are volume emissions provided?
`Scavenging_model`	Which scavenging model is applied? If `none`, the scavenging is not taken into account. Otherwise, the following model is applied: `constant` for constant scavenging coefficient, `belot` for the Belot model (of the form $a\,{p_0}^b$, where p_0 is the rain intensity in $\mathrm{mm\,h^{-1}}$) or `microphysical` for the scavenging model based on microphysical properties of species.
`Collect_dry_flux`	Are the dry deposition fluxes collected in order to post-process them if dry deposition is taken into account?
`Collect_wet_flux`	Are the wet deposition fluxes collected in order to post-process them if wet deposition is taken into account?
	`[data]`
`Data_description`	Path to the configuration file that describes input data.
`Horizontal_diffusion`	Horizontal diffusion coefficient in $\mathrm{m^2\,s^{-1}}$.
`Isotropic_diffusion`	If activated, horizontal diffusion is set equal to vertical diffusion.
	`[output]`
`Configuration_file`	Path to the configuration for the output saver.

5.5.2 Data Description: `polair3d-data.cfg`

This configuration file describes input data files (binary files). It is divided into sections: for deposition, for meteorological fields, etc. A section roughly looks like this:

```
[meteo]

Date_min: 2004-08-09
Delta_t = 10800.

Fields: MeridionalWind ZonalWind Temperature Pressure Rain CloudHeight Attenuation\
SpecificHumidity
Filename: /u/cergrene/a/ahmed-dm/TestCase-1.0/data/meteo/&f.bin

VerticalDiffusion: /u/cergrene/a/ahmed-dm/TestCase-1.0/data/meteo/Kz_TM.bin
```

It is assumed that all binary files start at the same date, and this date is `Date_min` (see dates formats in Section D.7). The time step is `Delta_t`, in seconds.

Then a list of fields is provided after `Fields`. These are fields that the model needs, and their names are determined by the model. Below, all fields required by the model (depending on its options) are listed. A generic path (full file name) is then provided (entry `Filename`). In this path, the shortcut '`&f`' refers to a field name. In the previous example, the full path to the temperature is `/u/cergrene/a/ahmed-dm/TestCase-1.0/data/Temperature.bin`. In the specific case of boundary conditions, the shortcut '`&c`' is replaced by x, y and z.

If a few fields are not stored in a file with a generic path, their specific paths can be provided after the entry `Filename`. This is the case for `VerticalDiffusion` in the previous example.

Note that:

1. entries `Fields`, `Filename` and additional paths *must* be at the *end of the section*, and *in this order*;
2. *at least one element* (possibly not a required field) must be provided to `Fields` and at least one element (possibly not a path) to `Filename`; for instance:

   ```
   Fields: ---
   Filename: --- # means no generic path.
   ```

   but:

   ```
   Fields:    # Illegal: one element required.
   Filename:     # Illegal: one element required.
   ```

In most sections, `Fields` is used to specify all chemical species involved in the process, e.g.:

```
[deposition]

Date_min: 2001-01-02
Delta_t = 10800.

Fields: O3 NO NO2 H2O2 HCHO PAN HONO SO2 HNO3 OP1 PAA ORA1
Filename: /u/cergrene/A/mallet/2001/data/dep-2005-01-19/&f.bin

ALD   /u/cergrene/A/mallet/2001/data/dep-2005-01-19/ALD-modified.bin
CO    0.002
```

Notice that `CO` is not associated with a path but with a numerical value. This is a feature: a binary file may be replaced with a numerical value. In this case, the field (in the example, CO deposition velocity) is set to a constant value (in every cell and at every time step). This works with any field, including meteorological fields (section `[meteo]`). This feature is often used to set constant boundary conditions.

In `polair3d-data.cfg`, several sections are required. Several sections have to be included only if given options are activated. In the following table, all possible sections are listed, with their entries.

Section	Entries	Comments
`[initial_condition]`	`Fields`, `Filename`	If initial conditions are activated (`With_initial_condition`).
`[boundary_condition]`	`Date_min`, `Delta_t`,`Fields`, `Filename`	If boundary conditions are activated (`With_boundary_condition`).
`[meteo]`	`Date_min`, `Delta_t`, `Fields`, `Filename`	Required fields are: `MeridionalWind` and `ZonalWind` if advection is activated, `VerticalDiffusion` if diffusion is activated, and `Temperature` and `Pressure` in case air density is taken into account.

[deposition]	Date_min, Delta_t, Fields, Filename	If deposition is activated (With_deposition).
[point_emission]	file	Path to the file which defines the point emissions (described below). If point emissions are activated (With_point_emissions).
[surface_emission]	Date_min, Delta_t, Fields, Filename	If surface emissions are activated (With_surface_emission).
[additional_surfa...]	Date_min, Delta_t, Fields, Filename	If surface additional emissions are activated (With_additional_surface_emission). This is mostly useful for biogenic emissions. Note that a species with additional surface emissions must have emissions in [surface_emission]. You might need to add the given species (say ISO) in section [surface_emission] with zero emissions (a line like ISO: 0).
[volume_emission]	Date_min, Delta_t, Nz, Fields, Filename	If volume emissions are activated (With_volume_emission). Nz is the number of levels in which pollutants are emitted.
[scavenging]	Fields	If the scavenging model is not set to "none" (Scavenging_model).

If there are point emissions, the point emission file used by the model is of the general type described Section 5.15.

5.5.3 Vertical Levels and Species

Vertical levels are defined in a single data file. They are defined by their interfaces. This means that the file contains Nz+1 heights, where Nz is the number of levels specified in the main configuration file. The concentrations are computed at layers mid-points.

Species are listed in the section [species] of a configuration file. In addition, some scavenging models needs extra data:

- The constant model requires a section [scavenging_coefficient] which contains a threshold of rain to apply scavenging (in $\mathrm{mm\,h^{-1}}$) and the name of the species with its associated scavenging coefficient (in $\mathrm{s^{-1}}$); for instance:

  ```
  [scavenging_coefficient]

  # Scavenging is applied above the following threshold over rain [mm / h].
  Scavenging_rain_threshold: 1.

  # Scavenging coefficient of the species: [s^{-1}]
  NO2 1.e-4       SO2 1.e-4
  ```

  Notice that if the previous lines are replaced by

  ```
  # Scavenging coefficient of the species: [s^{-1}]
  all    1.e-4
  ```

  the same scavenging coefficient will be used for all scavenged species.

- The `belot` model has the following expression $a\,{p_0}^b$ where coefficients a and b have to be provided for every species in a section `[belot]`; for instance:

```
[belot]

# Coefficients a and b for the Belot parameterization ($a * {p_0}^b$)
#  where po is the rain intensity  [mm / h].

# species       a       b
  all           1.e-05  0.8
```

- In case the `microphysical` model is used, Henry constants (in $\mathrm{mol\,L^{-1}\,atm^{-1}}$) and gas-phase diffusivities (in $\mathrm{cm^2\,s^{-1}}$) should be provided. Henry constants are listed in section `[henry]`; for instance:

```
[henry]

# Henry constant: [mol / L / atm]

O3      1.e-2   NO      2.e-3   NO2     1.e-2   H2O2    1.e5
HCHO    6.e3    ALD     15.     PAN     3.6     HONO    1.e5
SO2     1.e5    HNO3    1.e14   OP1     2.4e2   PAA     5.4e2
ORA1    4.e6    CO      1.e3    N2O5    1.e14
```

  Gas-phase diffusivities are provided in the same way in section `[diffusivity]`.

5.6 Polair3DChemistry

Model Polair3DChemistry is configured with three configuration files:

- Main configuration file: `polair3d.cfg` for RADM, `racm.cfg` for RACM, `racm2.cfg` for RACM2 and `cb05.cfg` for CB05
- Data description file: `polair3d-data.cfg` for RADM, `racm-data.cfg` for RACM, `racm2-data.cfg` for RACM2 and `cb05-data.cfg` for CB5
- Output saver file: `polair3d-saver.cfg` for RADM, `racm-saver.cfg` for RACM, `racm2-saver.cfg` for RACM2 and `cb05-saver.cfg` for CB05

and two data files (`levels.dat` and `species-racm.dat` for RACM). The main configuration files (`polair3d.cfg`) provide the paths to the four other files.

A configuration for Polair3DChemistry is an extension of the configuration for Polair3DTransport. In this section, the description is limited to Polair3DChemistry additional configuration. See Section 5.5 for the rest of the configuration.

5.6.1 Main Configuration File: `polair3d.cfg`

In addition to fields introduced in Section 5.5.1, the following fields are read by Polair3DChemistry:

	[options]
With_chemistry	Should chemistry occur?
With_photolysis	Should photolysis occur?
With_forced_concentrations	If activated, the concentrations of a few species are set to values read in files.
Source_splitting	If activated, source splitting is used within chemistry integration. Advection and diffusion fluxes are included in the chemistry integration as sources. This slightly increases the memory requirements but is recommended for numerical stability.
With_adaptive_time_step_for_gas_chemistry	With adaptive time stepping for gaseous chemistry?
Adaptive_time_step_tolerance	Tolerance for the adaptive time step.
Min_adaptive_time_step	Minimum for the adaptive time step.
Photolysis_option	If 1, photolysis rates are computed from tabulations. If 2, they are read from binary files created by preprocessing tools (JProc, FastJ ...).
Option_chemistry	Chemistry mechanism used in the simulation. You can choose among RACM, RACM2 and CB05.

5.6.2 Data Description: polair3d-data.cfg

In addition to the configuration described in Section 5.5.2, a section [photolysis_rates] may be required (if the chemical mechanism includes photolysis reactions). Photolysis rates depend on days, time angle, latitude and altitude. During the time integration, they are linearly interpolated in all cells.

Section	Entries	Comments
[photolysis_rates]	Date_min	Starting date of photolysis rates.
	Delta_t	Time step in days.
	Ndays	Number of steps.
	Time_angle_min	Starting time angle in hours.
	Delta_time_angle	Time angle step in hours.
	Ntime_angle	Number of time angles.
	Latitude_min	First latitude in degrees.
	Delta_latitude	Step along latitude in degrees.
	Nlatitude	Number of latitude steps.
	Altitudes	List of altitudes in meters at which photolysis rates are provided.
	Fields, Filename	Photolysis reaction names and the paths to the files in which photolysis rates are stored.

5.6.3 Vertical Levels and Species

Section 5.5.3 is relevant for Polair3DChemistry, and in particular the file giving the levels is exactly the same. As for species, a section [molecular_weight] lists the molecular weights (in $g\,mol^{-1}$) of *all* species. If photolysis reactions are involved, the section [photolysis_reaction_index] is required. This section provides all reaction names and their indices in the list of reactions. Below is an example for RACM.

```
[photolysis_reaction_index]

NO2      0       O3O1D    1       O3O3P    2       HONO     3
HNO3     4       HNO4     5       NO3NO    6       NO3NO2   7
H2O2     8       HCHOmol  9       HCHOrad  10      ALD      11
MHP      12      HOP      13      PAA      14      KETONE   15
GLYform  16      GLYmol   17      MGLY     18      UDC      19
ORGNIT   20      MACR     21      HKET     22
```

The previous section is quoted from `Polyphemus/processing/photochemistry/species-racm.dat` and is consistent with RACM (as implemented in Photochemistry – Section 6.2.1).

5.7 Polair3DAerosol

Polair3DAerosol is configured with three configuration files (`polair3d.cfg`, `polair3d-data.cfg` and `polair3d-saver.cfg`) and two data files (`levels.dat` and `species.dat`). The main configuration file (`polair3d.cfg`) provides the paths to the four other files.

A configuration for Polair3DAerosol is an extension of the configuration for Polair3DChemistry. In this section, the description is limited to Polair3DAerosol additional parameters. See Section 5.6 for the rest of the configuration.

5.7.1 Main Configuration File: `polair3d.cfg`

In addition to fields introduced in Section 5.6.1, the following fields are read by Polair3DAerosol.

	`[domain]`
`Bin_bounds`	The bounds of the diameter classes for aerosol species. Note that the classes are the same for each aerosol species.
	`[options]`
`With_initial_condition_aerosol`	Are initial conditions provided for given aerosol species? If not, initial concentrations are set to zero.
`With_boundary_condition_aerosol`	Are boundary conditions available for given aerosol species?
`With_pH`	Does the aerosol module returns cloud droplet pH?
`Lwc_cloud_threshold`	Liquid water content threshold above which a cloud is diagnosed in the cell.
`Fixed_aerosol_density`	Fixed aerosol density in $\mathrm{kg\,m^{-3}}$ used in the model.
`With_deposition_aerosol`	Is dry deposition taken into account for aerosol species?
`Compute_deposition_aerosol`	If set to yes, deposition velocities for aerosol species are computed with land data, otherwise they are read in files. Only needed if dry deposition is taken into account.
`With_point_emission_aerosol`	Are point emissions provided for aerosol species?
`With_surface_emission_aerosol`	Are emissions at ground provided for aerosol species?
`With_volume_emission_aerosol`	Are volume emissions provided for aerosol species?
`With_scavenging_aerosol`	Is there scavenging for aerosol species?
`With_in_cloud_scavenging`	Is there in cloud scavenging for aerosol species?
`Collect_dry_flux_aerosol`	Are the dry deposition fluxes are collected in order to postprocess them if dry deposition is taken into account?

Collect_wet_flux_aerosol	Are the wet deposition fluxes are collected in order to postprocess them if wet deposition is taken into account?

The bin bounds are presented as follow:

```
Bin_bounds:
# diameter of the particle classes in micrometers.
0.0 0.1 1 1.5 2 5
```

Note that these values are the bounds of the various diameter classes and that therefore there is one more value than there are classes.

5.7.2 Data Description: polair3d-data.cfg

In addition to the sections described in Section 5.6.2, some parameters may be necessary:

Section	Entries	Comments
[initial_condition_aerosol]	Fields, Filename	If initial conditions are activated (With_initial_condition_aerosol).
[boundary_condition_aerosol]	Date_min, Delta_t, Fields, Filename	If boundary conditions are activated (With_boundary_condition_aerosol).
[deposition_velocity_aerosol]	Fields, Filename	If deposition is activated (With_deposition_aerosol) and deposition velocities are not computed (Compute_deposition_aerosol set to no).
[point_emission_aerosol]	file	Path to the file which defines the point emissions. If point emissions are activated (With_point_emissions_aerosol).
[surface_emission_aerosol]	Date_min, Delta_t, Fields, Filename	If surface emissions are activated (With_surface_emission_aerosol).
[volume_emission_aerosol]	Date_min, Delta_t, Nz Fields, Filename	If volume emissions are activated (With_volume_emission_aerosol). Nz is the number of levels in which pollutants are emitted.

The point emissions, if needed, are given in files very similar to the point files described in Section 5.15 for gaseous species, except that there is only one species and one bin in each section [source]. The species and bin are given after the entry Species, in the form species_bin (for example PNO3_0). Only the types "continuous" and "puff" can be used.

5.7.3 Vertical Levels and Species

Section 5.6.3 is relevant for Polair3DAerosol. In addition, there is at least a section added in the file species.dat:

```
[aerosol_species]

PMD    PBC    PNA    PSO4   PNH4  PNO3   PHCL   PARO1
PARO2  PALK1  POLE1  PAPI1  PAPI2 PLIM1  PLIM2  PPOA   PH2O
```

5.8 Polair3DChemistryAssimConc

`Polair3DChemistryAssimConc` is dedicated for a state space formulation of the underlying dynamical model. The stochastic modeling is implemented for diverse applications such as data assimilation.

`Polair3DChemistryAssimConc` is configured with three configuration files (`polair3d.cfg`, `polair3d-data.cfg` and `polair3d-saver.cfg`) and two data files (`levels.dat` and `species.dat`). The four files other than the main configuration file (`polair3d.cfg`) are the same as those for `Polair3DChemistry`. The main configuration file is an extension of that of `Polair3DChemistry`.

The additional parameters are:

	`[state]`
`Species`	List of species included in model state vector. All species must be on the same line.
`Levels`	List of vertical levels of model domain included in model state vector. All levels must be on the same line.
	`[data_assimilation]`
`Error_covariance_model`	Stochastic model for model and background error covariance. With option set to `Balgovind`, the corresponding error covariance matrix is calculated using Balgovind correlation function; with option set to `diagonal_constant`, the corresponding error covariance matrix is a diagonal matrix of which the diagonal elements are error variances.
`Background_error_variance`	Error variance for background concentrations. The unit for the option value is $\mu g\, m^{-3}$.
`Balgovind_scale_background`	Balgovind scale for background error covariance. The model grid interval is chosen to be the unit for option values.
`Model_error_variance`	Error variance for model simulations (in $\mu g\, m^{-3}$).
`Balgovind_scale_model`	Balgovind scale for model error covariance. The model grid interval is chosen to be the unit for option values.

The data file is the same as in Section 5.6.2, the species and levels files are the same as those presented in Section 5.6.3.

5.9 CastorTransport

5.9.1 Main Configuration File: `castor.cfg`

Model `CastorTransport` is based on IPSL model Chimere. Its option are provided in a configuration file:

	`[domain]`
`Date_min`	Starting date in any legal format (see Section D.7). The date can therefore include seconds.
`Delta_t`	Time step in seconds.
`Nt`	Number of iterations of the simulation (integer).
`x_min`	Abscissa of the center of the lower-left cell. Provide a longitude (in degrees) or, in case Cartesian coordinates are chosen, an abscissa in meters.

`Delta_x`	Step length along x, in degrees (longitude) or in meters (for Cartesian coordinates).
`Nx`	Number of cells along x (integer).
`y_min`	Ordinate of the center of the lower-left cell. Provide a latitude (in degrees) or, in case Cartesian coordinates are chosen, an ordinate in meters.
`Delta_y`	Step length along y, in degrees (latitude) or in meters (for Cartesian coordinates).
`Ny`	Number of cells along y (integer).
`Nz`	Number of vertical levels (integer).
`Vertical_levels`	Path to the file that defines vertical levels interfaces. This field is read but is not used.
`Species`	Path to the file that defines involved species and their chemical properties.
	`[options]`
`With_transport`	Is transport taken into account?
`With_initial_condition`	Are initial conditions used?
`Interpolated_initial_condition`	If set to yes, initial conditions are interpolated from boundary conditions, otherwise they are read in binary files.
`With_boundary_condition`	Are boundary conditions provided?
`With_deposition`	Is deposition taken into account?
`With_volume_emission`	Are volume emissions taken into account?
	`[data]`
`Data_description`	Path to the configuration file that describes input data.
	`[output]`
`Configuration_file`	Path to the configuration for the output saver.

5.9.2 Data Description: `castor-data.cfg`

The data description is very similar to that of Polair3DTransport (see Section 5.5.1), except that the data can be different.

Section	Entries	Comments
`[initial_condition]`	`Fields`, `Filename`	If initial conditions are activated (`With_initial_condition`) and not interpolated (`Interpolated_initial_condition` set to no).
`[boundary_condition]`	`Fields`, `Filename`	If boundary conditions are activated (`With_boundary_condition`).
`[meteo]`	`Date_min`, `Delta_t`, `Fields`, `Filename`	Required fields are: `Temperature`, `Pressure`, `Altitude`, `AirDensity`, `MeridionalWind`, `ZonalWind` and `VerticalDiffusion`.
`[deposition]`	`Date_min`, `Delta_t`, `Fields`, `Filename`	If deposition is activated (`With_deposition`).

[volume_emission]	Date_min, Delta_t, Nz, Fields, Filename	If volume emissions are activated (With_volume_emission). Nz is the number of levels in which pollutants are emitted.

5.9.3 Vertical Levels and Species

A file containing vertical levels similar to the one for "Polair3D" models is read but is not useful. Give any such file.

Species file has two sections:

- [species] which contains all species managed by the simulation.
- [species_ppm] which contains all species for which an upwind scheme is not used.

5.10 CastorChemistry

Model CastorChemistry is derived from CastorTransport and all data presented in Section 5.9 are necessary for this model too. In addition some other parameters are needed.

5.10.1 Main Configuration File: castor.cfg

[options]	
With_chemistry	Is chemistry taken into account?
[chemistry]	
Reaction_file	Data file containing the reactions.
Stoichiometry_file	Data file containing the stoichiometry of the reactions.
Photolysis_file	Data file containing the photolysis reactions.
Rates_file	Data file containing the reaction rates.

5.10.2 Data Description and Species

Data Description The only difference with what is described in Section 5.9.2 is that more meteorological fields are necessary (see Section 2.6.4).

Species The species file is exactly the same as the one presented in Section 5.9.3.

5.10.3 Chemistry Files

When model CastorChemistry is used, four files are necessary to describe the chemistry:

- Reaction_file which gives the reactions between the species managed in the format

  ```
  2   CO   OH     2  HO2  CO2
  ```

  The first column gives n the number of species reacting, the n following columns give the name of the species reacting, the following columns are the number and names of the species resulting from the reaction.

- Stoichiometry_file

- Photolysis_file
- Rates_file

5.11 PlumeInGrid

The base files for plume-in-grid model are PlumeInGrid.hxx and PlumeInGrid.cxx. Basically, it is a model, but it can be used as a driver. The plume-in-grid model uses both an Eulerian model and a Gaussian Puff model. It processes major point emissions first with the Puff model, then feeds the puffs back to the Eulerian model when their size is large enough. Apart from this special treatment of point sources, the Eulerian simulation is performed as if BaseDriver were used. The Eulerian model can be, for example, Polair3DAerosol, Polair3DChemistry, or CastorChemistry. For the time being, only Polair3DChemistry with RACM mechanism can be used for reactive cases.

The plume-in-grid model uses basic configuration files for the Eulerian model, with some changes that are described below, and the file puff.cfg for options for the Gaussian puff model.

5.11.1 Main configuration file

In the main configuration file, there is a new section named [gaussian] where the name of the configuration file for the puff model has to be provided (field file_gaussian).

5.11.2 Data description file

In the data configuration file, there is a new section [gaussian_meteo] that provides more meteorological fields than the ones used for Eulerian model:

1. Fields LowCloudiness, MediumCloudiness, HighCloudiness and SolarRadiation will be used to compute stability class. It is used only with Briggs parameterization for standard deviations. In other cases, put whatever value you want for those fields (they are still read but not used).
2. Fields FirstLevelWindModule, FrictionModule, BoundaryHeight and LMO provide respectively the friction velocity, the boundary layer height and the Monin-Obukhov length. They are always read but used only in case similarity theory is used to compute standard deviations. In other cases, put any value for those fields.

In addition, there is a new section [plume-in-grid] that provides several information detailed below.

[plume-in-grid]	
Horizontal_coefficient	Coefficient by which to multiply σ_y to obtain the puff horizontal size (by default: 4).
Vertical_coefficient	Coefficient by which to multiply σ_z to obtain the puff vertical size (by default: 4).
With_reinjection_time	Is puff reinjection forced after a given time? (if set to "no", the reinjection criterion is based on the puff horizontal size).
With_interpolation	Are meteorological data taken at the center of the puff cell ("no") or interpolated at the puff center ("yes")?

`Reinjection_time`	Reinjection time in seconds after the puff emission (used only if `With_reinjection_time` is set to "yes").
`Injection_method`	Puff feedback method into the Eulerian model:"column" for an injection on one column of cells, "integrated" for an injection proportional to the puff quantity in all neighboring cells.
`With_chemistry_feedback`	Is there some feedback of the chemical products in the Eulerian cell at each time step ("yes"), or does everything stay in the puff until reinjection ("no")?

Note: The option `With_chemistry_feedback` allows to save some computational time, since the chemical interaction between the background species in a cell and all puffs within this cell is computed only once, and not for each puff separately. **However, it is not advised to use this option** for the time being, since its stability is not guaranteed.

Note that there is no point emission for the Eulerian model, since the source file is directly read by the Gaussian puff model (it is provided in the Gaussian puff configuration file). Hence, if you set the option `With_point_emission` to "yes" and provide the same configuration file for point source, the source file will be read twice and taken into account by both models.

Finally, a section `[ground]` provides some information about the land use coverage. This is used by the Briggs parameterization: it uses the urban formulae when the cell contains more than a given proportion of the urban luc. Instead of a binary file, the keywords "rural" or "urban" can be provided, to use the corresponding Briggs formulae.

`[ground]`	
`LUC_file`	Path to the binary file that describes land use cover or `rural`, or `urban`.
`Urban_index`	Index of the urban areas in land categories (0 for usgs, and 13 for glcf).
`Urban_proportion`	Proportion of urban LUC in a cell to use the urban formulae. Default: 0.25.

5.11.3 Puff configuration file: `puff.cfg`

In addition, the Gaussian puff model needs the usual configuration files. However, few of their information are actually used, since most information are directly provided by the plume-in-grid model.

The most important information given in `puff.cfg` are the time step `Delta_t`, and the source information: `File_puff` is the path to the source file (point sources to be treated by the Gaussian model), and `Delta_t_puff` is the time step between two puff emissions.

There are two constraints for these time step values:

1. Time step of the Gaussian model must not be smaller than time step between two puff emissions. The plume-in-grid model checks this, and sets `Delta_t_puff` equal to the value of `Delta_t` if necessary.

2. Time step of the Gaussian model is used in an inner-loop of the Eulerian time-loop. Hence, it must not be greater than Eulerian time step. The number N of iterations for the Gaussian

model performed at each iteration of the Eulerian model is therefore computed as:

$$N = \max\left(\lfloor\frac{\Delta t_{\text{Eulerian}}}{\Delta t_{\text{Gaussian}}}\rfloor, 1\right) \tag{5.1}$$

In short, it is ensured that $\Delta t_{\text{puff}} \geq \Delta t_{\text{Gaussian}}$ and $\Delta t_{\text{Gaussian}} \leq \Delta t_{\text{Eulerian}}$ where $\Delta t_{\text{Gaussian}}$ is the time step of the Gaussian model (`Delta_t` in file `puff.cfg`), Δt_{puff} is the time step between two puff emissions, and $\Delta t_{\text{Eulerian}}$ is the time step of the Eulerian model, given in the main configuration file.

The plume-in-grid model needs some other information in file `puff.cfg`: it reads all options and parameterizations. Note that with plume-in-grid, it is advised to set the option `With_increasing_sigma` to 'yes'.

For plume-in-grid with chemistry, the option `With_chemistry` has to be set to 'yes' both in the main configuration file and in `puff.cfg`. The option `With_puff_interaction` normally has to be set to 'yes', in order to take into account the chemical interaction between two puffs. It can be set to 'no' to save computational time, but it is not advised to do so.

Other information is read but not used. Scavenging, deposition and radioactive decay can be used in the Gaussian model. For scavenging and deposition to be used, the option has to be set to 'yes', both in the Gaussian puff model and in the Eulerian model, in order to ensure consistency between the two models. If deposition in the Gaussian model is used, the Chamberlain deposition has to be used, instead of the Overcamp model.

There is no need to provide a meteorological file, since these data are fed to the puff model by the plume-in-grid model. The species file is still read. It is advised to use the same as the species file for Eulerian model. The names of the levels file and saver file are still read but the files are not used.

5.12 StationaryModel

This model is used to perform a simulation at local scale, with an Eulerian model. The model mostly creates an interface between the driver (normally `BaseDriver`) and an underlying Eulerian model, such as `Polair3DChemistry`. At each time step of `StationaryModel` ("simulation time step"), iterations of the underlying model are performed with a much smaller time step ("internal time step") until convergence is reached. The driver moves `StationaryModel` from one simulation time step to the next one.

Convergence is checked by computing the norm of the difference between the concentrations before and after the iteration (1-norm, 2-norm or infinity-norm, see below), normalizing it with the mean or the maximum of the concentration before and by comparing it to a convergence criterion.

The output on screen for each iteration gives whether or not the inner-loop converged and , if it did, after how many iterations. If the number of iterations is too small or too big, this means that the way to check convergence is not adapted. Try changing the norm used, the method or the convergence criterion.

Both models (`StationaryModel` and the underlying model) are built with the same configuration file(s). Except for a few points described below, the configuration file must be the

one for the underlying model (see the Section describing it). The differences are only in section `[domain]` and two additional sections (`[stationary]` and `[convergence]`):

	`[domain]`
`Nt`	Maximum number of iterations of the underlying model. **This is different from what is normally defined here.**
`Delta_t`	Time-step of the internal loop ("internal time step"). **This is different from what is normally defined here.**
	`[stationary]`
`Nt`	Number of time steps for the simulation.
`Delta_t`	Time-step of the simulation (in seconds).
	`[convergence]`
`Norm`	Norm used to check convergence: `one`, `two` or `infinity`.
`Method`	Method used to normalize the norm: `mean` or `max`.
`Epsilon`	Convergence criterion (for instance `1.e-4`).

5.13 LagrangianTransport

The LagrangianTransport model was implemented according to Wendum [1998]. The implementation is still primary and does not take into account such phenomenon as the scavenging, the wet or dry deposition, the boundary layer effect etc...

It has successfully been tested and evaluated with the ETEX dispersion case where these parameterizations are not significant.

Its use is illustrated in `processing/lagrangian-stochastic`. It is there configured with three configuration files:

- `lagrangian-stochastic.cfg`,
- `lagrangian-stochastic-data.cfg`,
- `lagrangian-stochastic-saver.cfg`,

and two data files:

- `levels.dat`,
- `point_emission.dat`.

5.13.1 Main Configuration File: `lagrangian-stochastic.cfg`

The main configuration file `lagrangian-stochastic.cfg` gives information on the domain definition and the options of the simulation:

	`[domain]`
`Date_min`	Starting date in any legal format (see Section D.7). The date can therefore include seconds.
`Delta_t`	Time step in seconds.
`Nt`	Number of timesteps (integer).
`x_min`	Abscissa of the center of the lower-left cell. Provide a longitude (in degrees).

`Delta_x`	Step length along x, in degrees (longitude).
`Nx`	Number of cells along x (integer).
`y_min`	Ordinate of the center of the lower-left cell. Provide a latitude (in degrees).
`Delta_y`	Step length along y, in degrees (latitude).
`Ny`	Number of cells along y (integer).
`Nz`	Number of vertical levels (integer).
`Vertical_levels`	Path to the file that defines vertical levels interfaces.
`Species`	Path to the file that defines involved species.
	`[species]`
	List of species.
	`[options]`
`With_air_density`	If activated, vertical wind is diagnosed from $\mathrm{div}(\rho V) = 0$ where ρ is the air density and V the wind, and the diffusion term is $\mathrm{div}\left(\rho K \nabla \frac{c}{\rho}\right)$ where c is the concentration and K is the diffusion matrix. If this option is not activated, it is assumed that ρ is constant and therefore disappears from the previous equations.
`With_point_emission`	Are point emissions provided? Should be set to `yes` because there is now no other way to introduce particles in a Lagrangian simulation!
	`[data]`
`Data_description`	Path to the configuration file that describes input data.
`Horizontal_diffusion`	Horizontal diffusion coefficient in $\mathrm{m}^2\,\mathrm{s}^{-1}$.
`Gaussian_kernel_horizontal_diffusion`	Horizontal diffusion coefficient used in the computation of the gaussian kernel distribution in $\mathrm{m}^2\,\mathrm{s}^{-1}$.
	`[output]`
`Configuration_file`	Path to the configuration for the output saver.

5.13.2 Data Description: `lagrangian-stochastic-data.cfg`

The data configuration file is very simple because the model LagrangianTransport takes into account very few parameterizations.

	`[meteo]`
`Date_min`	Starting time of the meteo data files (see dates formats in Section D.7).
`Delta_t`	Timestep of the meteo data files (in seconds).
`Fields`	Required fields are: `MeridionalWind` and `ZonalWind`, and `Temperature` and `Pressure` in case air density is taken into account.
`Filename`	Generic path where the shortcut '`&f`' refers to a field name defined in `Fields`.
`VerticalDiffusion`	File name for `VerticalDiffusion`.

	[point_emission]
file	Path to the file which defines the point emissions (described below).
Delta_t_particle_emission	Time interval between the emission of two particles (in seconds).

5.13.3 Vertical Levels and Point Emission

Vertical levels are defined the same way as in 5.5.3. As for point emission, its related file is of the general type described in Section 5.15.

5.13.4 Noteworthy Remarks about Output Saving

The simulation outputs can be saved with the unit saver types related to the `SaverUnitDomain` (see 4.9.2) or `SaverUnitPoint` (see 4.9.7). The following types should therefore be available: `domain`, `domain_ensemble_forecast`, `domain_ensemble_analysis`, `indices_list` and `coordinates_list`. Take notice that the option `Averaged` is not supported: it should then always be set to `no`.

When using `SaverUnitDomain`, you might be surprised by the time needed to complete the saving of a given timestep. This deserves some further explanations.

Unlike Eulerian models, Lagrangian models do not compute the concentration in every cells of the domain at each timestep. This calculation has to be performed besides the Lagrangian algorithm as an explicit projection from the Lagrangian particles onto the domain mesh. Such a projection can be very expensive in CPU time depending on the particle type and the mesh size.

The particles that are now implemented in Polyphemus are Gaussian kernels whose projection is particularly costful because each particle is considered to contribute to each cell concentration. This was inherited from the DIFPAR algorithm, so it is reasonable to expect significant improvement in versions to come.

As a consequence, you are invited to consider saving the results over the whole domain as a scarce ressource whereas saving results at a list of points has to be favoured whenever it makes sense. Of course, the number of saved levels is also to be decided carefully.

5.14 Lagrangian Particles

The `LagrangianTransport` model can transport different types of particles who might differ for instance, by the way their displacement is computed given meteorological velocity fields or by the way their mass distribution is modelled. You can change the type of particle you want in `processing/lagrangian-stochastic/lagrangian-stochastic.cpp` when declaring your model:

```
typedef LagrangianTransport<real, ParticleDIFPAR_Horker<real> >
  ClassModel;
```

`ParticleDIFPAR_Horker` was set up above. This particle model is implemented in the related files you should easily find in `include/models`. You can also decide to develop your own particle model. In such a case, derive it from the `BaseParticle` class (or from other existing classes). They are all located in `include/models`.

Up to now, following Wendum [1998], we implemented two types of particles, both using the Gaussian kernel model for mass distribution.

5.14.1 ParticleDIFPAR_Horker

The Horker formulation is so simple that there are no parameter to be set. Notice that the fields `Horizontal_diffusion` and `Gaussian_kernel_horizontal_diffusion` defined in `processing/lagrangian-stochastic/lagrangian-stochastic.cfg` are irrelevant for this particle model.

5.14.2 ParticleDIFPAR_FokkerPlanck

Like Wendum [1998], we considered the horizontal diffusion splits between the horizontal turbulence used in the displacement computation (named simply `Horizontal_diffusion`) and the horizontal diffusion used in the mass distribution (named `Gaussian_kernel_horizontal_diffusion`). In the ETEX case of Wendum [1998], the first field was set to $25000\text{m}^2\,\text{s}^{-1}$ whereas the the second one was assigned $50000\text{m}^2\,\text{s}^{-1}$.

5.15 Point Emission Management

The management of point emissions for gaseous species is common to several models. In particular, Polair3D, GaussianPlume (Section 5.1.2), GaussianPuff (Section 5.3.2), PlumeInGrid and LagrangianTransport (Section 5.13) use the common point emission manager. The configuration file needed for these emissions, as well as the various types of point emissions available, are described in this section.

Important note: The common point emission manager is only available for gaseous species, for the time being. The aerosol species are still managed by each model separately. For point emission files used with GaussianPlume_aer and GaussianPuff_aer, please refer to the Sections 5.2.2 and 5.4.2 respectively. For point emission files used with Polair3DAerosol, please refer to the Section 5.7.2.

The file for gaseous point emissions has to contain a section `[source]` for each point emission, with the following features:

- its location: `Abscissa` and `Ordinate` are given in degrees (or in meters, in case Cartesian coordinates are chosen) and `Altitude` is the vertical height in meters. For Eulerian models, the emission is released in the cell containing the location of the point emission.
- the list of emitted species is filled after `Species`.
- the type `Type` may be `continuous`, `puff` for instantaneous release, `temporal` for continuous emissions varying in time or `continuous_line` for continuous line emissions.

5.15.1 Continuous emissions

The `continuous` emission is described with the following entries:

`Date_beg`	The date at which the emission starts. The date must be in a format described in Section D.7.
`Date_end`	The date at which the emission ends. The date must be in a format described in Section D.7.
`Rate`	The list of rates (one per emitted species) in mass unit per seconds.
`Velocity`	The stack exit velocity (m/s)

`Temperature`	The source temperature (Celsius degrees)
`Diameter`	The source diameter (m)

The source velocity, temperature and section are read, and used for plume rise calculation if the option is activated (namely in Gaussian models). If they are not known, put zero. The diameter may also be used to compute the initial horizontal extent of the emitted plume. The configuration file for point emissions may contain a section looking like this:

```
[source]

Abscissa: 5.2
Ordinate: 48.5
Altitude: 10.

Species: NO NO2

Type: continuous
Rate: 1. 1.5
Date_beg: 2001-04-22_00-05
Date_end: 2001-04-22_00-07

Velocity: 0.
Temperature: 0.
Diameter: 0.2
```

5.15.2 Puff emissions

The `puff` emission is described with the following entries:

`Date`	The date at which the puff is emitted. The date must be in a format described in Section D.7.
`Quantity`	The list of quantities (one per emitted species) in mass unit.

The configuration file for point emissions may contain a section looking like this:

```
[source]

Abscissa: 10.3
Ordinate: 48.
Altitude: 80.

Species: SO2

Type: puff
Quantity: 1.
Date: 2001-04-22_00-05

Velocity: 0.
Temperature: 0.
Diameter: 0.2
```

5.15.3 Temporal emissions

The `temporal` emission is the same as a `continuous` emission, but temporal factors are applied to the emission rate in order to account for temporal variations. In addition to the continuous emission entries, the following entries are required:

`Date_min_file`	The date at which the temporal factor file starts. The date must be in a format described in Section D.7, and must be lower than, or equal to, the emission beginning date.
`Delta_t`	The time step (in seconds) between two temporal factors.
`TemporalFactor`	The name of the binary file where the temporal factors are read.

At each simulation time step, the emission rate is multiplied by the temporal factor read in the file. The index of the temporal factor is given by the file beginning date and time step, as well as the date at the current simulation time step. The temporal factor file is a binary file, containing the list of factors, from the beginning date (in float type).

The configuration file for point emissions may contain a section looking like this:

```
[source]

Abscissa: 10.3
Ordinate: 48.
Altitude: 80.

Species: SO2

Type: temporal
Rate: 1.
Date_beg: 2001-04-22_00-05
Date_end: 2001-04-23_00-00
TemporalFactor: hourly_factor.bin
Date_min_file: 2001-04-01
Delta_t: 3600.
```

5.15.4 Continuous line emission

The `continuous_line` emission is described with the following entries:

`Date_beg`	The date at which the emission starts. The date must be in a format described in Section D.7.
`Date_end`	The date at which the emission ends. The date must be in a format described in Section D.7.
`Rate`	The list of rates (one per emitted species) in mass unit per second per meter.
`Coordinate_file`	Path of the coordinate file (which replaces the Abscissa, Ordinate and Altitude entries of the continuous point emission, described in Section 5.15.1). It has the same syntax as the data file `line-emission.dat` described in Section 3.9.1.

The configuration file for point emissions may contain a section looking like this:

```
[source]
```

```
Species: Iodine Caesium

Type: continuous_line
Date_beg: 2001-01-01_01-00-00
Date_end: 2001-01-01_04-00-00
Rate: 56.5 27.2

Coordinate_file: line-emission.dat
```

Chapter 6

Modules

6.1 Transport Modules

6.1.1 AdvectionDST3

Module `AdvectionDST3` is the transport module associated to advection for `Polair3D`. It is based on a third-order "direct space-time" scheme with a Koren-Sweby flux limiter. The data needed are the wind components and boundary conditions if they are available.

Please note that Courant-Friedrichs-Lewy (CFL) condition is not verified and that the user should choose the mesh dimensions and the time-step of simulations very carefully. In order to enforce the CFL, you may use module `SplitAdvectionDST3` instead.

6.1.2 SplitAdvectionDST3

Module `SplitAdvectionDST3` is the same as `AdvectionDST3` except that

- it uses directional splitting;
- it performs automatic subcycling in order to satisfy the CFL.

It is the advection module used in every program `driver/*.cpp`, except the ones for data assimilation.

6.1.3 GlobalAdvectionDST3

Module `GlobalAdvectionDST3` is the same as `AdvectionDST3` for global scale (boundary conditions are not used).

6.1.4 DiffusionROS2

Module `DiffusionROS2` is the transport module associated to diffusion for `Polair3D`. It is based on a second-order Rosenbrock method. Fortran routines are used to perform all numerical computations.

6.1.5 GlobalDiffusionROS2

Module `GlobalDiffusionROS2` is the same as `DiffusionROS2` for global scale.

6.1.6 TransportPPM

Module `TransportPPM` is the numerical solver for transport used in Castor model. It uses piecewise parabolic method (PPM) for advection but can also use an upwind scheme for some species.

In the species file associated with castor there are two sections: `[species]` and `[ppm_species]`. For all species in `[species]` but not in `[ppm_species]` an upwind scheme will be used.

6.2 Chemistry Modules

6.2.1 Photochemistry

Module `Photochemistry` is the most common photochemical module used with Polair3D. It implements three chemical mechanisms: RACM (Stockwell et al. [1997]), RACM2(Goliff and Stockwell [2008]) and CB05(Yarwood et al. [2005]). It uses a second-order Rosenbrock method for time integration. Computations are performed by Fortran routines (automatically generated by the chemical preprocessor SPACK) and a C++ program is used as a frame to launch all these calculations.

It only deals with gaseous species. Information about species and reactions is given below.

Chemical mechanisms	no of species	no of reactions
`RACM`	72	237 including 23 photolysis
`RACM2`	113	349 including 34 photolysis
`CB05`	52	155 including 23 photolysis

The *units* of input data (e.g., initial condition, boundary condition etc.) should be given as $\mu g/m^3$ in using `Photochemistry` module.

6.2.2 ChemistryRADM

Module `ChemistryRADM` is quite similar to `Photochemistry`. RACM has actually been derived from RADM.

RADM manages 61 species, 157 reactions involving those species and 21 photolysis reactions.

The *units* of input data (e.g., initial condition, boundary condition etc.) should be given as $\mu g/m^3$ in using `ChemistryRADM` module.

6.2.3 ChemistryCastor

Module `ChemistryCastor` is the default chemical module for `Castor`. It involves 44 species and 118 reactions. It is based on several data files which must be provided: `Reaction_file`, `Stoichiometry_file`, `Photolysis_file` and `Rate_file`.

6.2.4 Decay

This chemistry module is used for species (gaseous or particulate) which have a radioactive or biological decay, that is to say a natural decrease in their concentrations over time. It requires two more options in the configuration file (`polair3d.cfg`).

`[options]`	
`With_time_dependence`	If set to yes, the value of the half-life time for each species depend on the time of the day.

`With_filiation_matrix`	If set to yes, decay and filiation are represented by a matrix.

Note that `With_time_dependence` and `With_filiation_matrix` cannot be both set to yes at the same time.

Use of One Value of Decay The first possibility is that each species has a half-life time which is given in `species.dat`. In that case `With_time_dependence` and `With_filiation_matrix` are both set to no. The variation of concentration due to decay only is described in equation (6.1) where $T_{1/2}$ is the species half-life time in days and t_0 is a reference time. If a species has no decay, its half-life time is set to 0, and this is interpreted by `Decay` as the fact that concentration does not vary due to decay.

$$C(t) = C(t_0) \exp\left(-\frac{(t - t_0)\ln 2}{T_{1/2}}\right) \tag{6.1}$$

The parameters needed are provided in `species.dat` as follow.

```
[species]

Sp1 Sp2 Sp3 Sp4

[aerosol_species]

Aer1 Aer2

[half_life]

# Half-lives in days, put 0 for species without decay.
Sp1 300
Sp2 216
Sp3 0
Sp4 41

[half_life_aerosol]

# Half-lives in days, put 0 for species without decay.
Aer1   250
Aer2   120
```

Use of Two Values of Decay Another option is that each species has two values of $T_{1/2}$, one for the day and one for the night. This is in particular the case for species which have a biological effect. As before, for a species without decay, both half-life times are set to 0. The equation involved is very similar to equation (6.1), except that the value of $T_{1/2}$ can vary. In that case `With_time_dependence` is set to yes and `With_filiation_matrix` to no.

The parameters needed are provided in `species.dat`.

```
[species]

Sp1 Sp2 Sp3 Sp4
```

```
[aerosol_species]

Aer1 Aer2

[half_life_time]

# Half-lives in days, put 0 for species without decay.
# First value for day, second for night.

Sp1 300 500
Sp2 216 300
Sp3 0 0
Sp4 41 72

[half_life_time_aerosol]

# Half-lives in days, put 0 for species without decay.
# First value for day, second for night.
Aer1: 250 350
Aer2: 120 180
```

`Decay` tests whether it is day or night and chooses the value of half-life time to use.

Use of a Filiation Matrix The last solution is that a single matrix (called filiation matrix) is specified for all gaseous species (and one for all aerosol species), which takes into account both decay and the fact that a species can react to form other species. As a result, the evolution of the concentration due to decay only is described in equation (6.2). In that case `With_time_dependence` is set to no and `With_filiation_matrix` to yes.

$$C^{n+1}(x,y,z) = AC^{n}(x,y,z) \tag{6.2}$$

where A is the $s \times s$ *filiation matrix* and $C^n(x,y,z) = \begin{pmatrix} \mathrm{c}_0^n(x,y,z) \\ \dots \\ \mathrm{c}_i^n(x,y,z) \\ \dots \\ \mathrm{c}_{s-1}^n(x,y,z) \end{pmatrix}$ with $\mathrm{c}_i^n(x,y,z)$ the concentration of species i at time-step n in point of coordinate (x,y,z) and s the number of species involved.

The parameters are specified as follows, in `species.dat`

```
[species]

Sp1 Sp2 Sp3 Sp4

[aerosol_species]

Aer1 Aer2

[filiation_matrix]
```

```
File: matrix.dat

[filiation_matrix_aerosol]
File: matrix_aer.dat
```

The $s \times s$ filiation matrix is specified in file matrix.dat as below :

```
0.7 0.05 0 0.1
0 0.8 0.1 0.05
0.1 0.1 0.6 0.1
0.15 0 0.1 0.7
```

The matrix for aerosol species is very similar to the one for gaseous species, except that its size is $s_a \times s_a$ where s_a is the number of aerosol species.

6.3 Aerosol Modules

6.3.1 Aerosol_SIREAM_SORGAM

For `Aerosol_SIREAM_SORGAM` to work, you have to install `ISORROPIA` (see Section 1.3.5).

Aerosol_SIREAM_SORGAM This aerosol module is used for gas and aerosol species for general purposes as air quality modeling and risk assessment. The gas chemistry is solved with one of three chemical mechanisms: the RACM (Stockwell et al. [1997]), the RACM2(Goliff and Stockwell [2008]) and the CB05(Yarwood et al. [2005]. The aerosol dynamics is solved by the SIREAM model (Debry et al. [2007]). When a cloud is diagnosed in one cell of the domain, then instantaneous aerosol activation is assumed and the SIREAM model is replaced by the VSRM cloud chemistry model (Fahey and Pandis [2003]).

The number of aerosol bins is directly inferred from the number of bounds provided by the `Bin_bounds` option in main configuration file (`polair3d.cfg`). Further options are required in this configuration file.

	`[options]`
`With_pH`	Does the aerosol module returns cloud droplet pH?
`Scavenging_model`	Which below cloud scavenging model is used?
`Lwc_cloud_threshold`	Liquid water content threshold for clouds.
`With_coagulation`	Is coagulation taken into account?
`With_condensation`	Is condensation taken into account?
`With_nucleation`	Is nucleation taken into account?
`Fixed_aerosol_density`	Fixed aerosol density in $\mathrm{kg\,m^{-3}}$ used in the module.
`aqueous_module`	Which aqueous module is used (`none`, `simple` or `VSRM`)?
`With_in_cloud_scavenging`	Is in-cloud scavenging taken into account?
`With_heterogeneous_reactions`	Are heterogeneous reactions taken into account?
`With_kelvin_effect`	Is Kelvin effect taken into account?
`Dynamic_condensation_solver`	Which solver is used for dynamic condensation (`etr`, `ros2` or `ebi`)?
`Fixed_cutting_diameter`	Fixed cutting diameter in µm.
`Sulfate_computation`	Which method is used to solve sulfate condensation (`equilibrium` or `dynamic`)?

`Redistribution_method`	Which redistribution method is used (`number-conserving` or `interpolation`)?
`Nucleation_model`	Which nucleation model is used (`binary` or `ternary`)?
`With_fixed_density`	Is aerosol density fixed in the module?
`Wet_diameter_estimation`	Which method is used to compute aerosol wet diameters (`Gerber` or `Isorropia`)?
`Thermodynamics_module`	Which thermodynamics module is used in bulk equilibrium (`isorropia` or `eqsam`, see below)?
`Number_of_forks`	Number of CPU to use (for more informations, consult the section about Multi-threading in 6.3.2).

The liquid water content threshold is the amount of liquid water in the air above which a cloud is diagnosed in the cell.

This chemistry module returns the cloud droplet pH, this means that `With_pH` can be set to yes, and that `microphysical-pH` scavenging model can be used. Otherwise choosing the `microphysical-pH` scavenging model may result in crash or errors.

Note that options `With_pH`, `Lwc_cloud_threshold` and `Fixed_aerosol_density` are used by both model and module. That is to say the fixed aerosol density is the same in the model as in the module.

The fixed cutting diameter has to be given as an aerosol diameter in µm. Aerosol bins below that diameter are assumed at equilibrium, and those above that diameter are not considered at equilibrium. The criteria is the comparison between the fixed cutting diameter and the bin bounds. The aerosol bin whose bounds are surrounding the fixed cutting diameter is included in the equilibrium bins.

Dynamic condensation is intended for aerosol bins which are not at equilibrium, and therefore time resolved mass transfer has to be computed for them. The solver for dynamic condensation may be set to either `etr` or `ros2` or `ebi`. The `etr` solver is an Explicit Trapezoidal Rule second order algorithm, the `ros2` solver is the Rosenbrock implicit second order scheme (Rosenbrock [1963]), and `ebi` is an Euler Backward Iterative scheme. Each of these solvers usually needs some numerical parameters, these are gathered in the Fortran include file `paraero.inc`.

Option `With_kelvin_effect` only affects dynamic bins.

Among aerosol species, sulfate condensation may have a different treatment. If `Sulfate_computation` is set to `equilibrium` then its treatment is equivalent to other species for both equilibrium and dynamic bins. But if it is set to `dynamic` then sulfate condensation is time resolved for all bins, using an analytic solution of mass transfer equations. This method is implemented in the `sulfdyn.f` Fortran routine.

As dynamic condensation is solved with a Lagrangian scheme, a redistribution process over the fixed aerosol size grid has to be performed at the end of condensation. Two methods are possible: `number-conserving` or `interpolation`. The former conserves the relationship between mass and number concentration in each bin, the latter relaxes this relationship.

Available nucleation models are either `binary` nucleation H_2SO_4–H_2O (Vehkamäki et al. [2002]) or `ternary` nucleation H_2SO_4–H_2O–NH_3 (Arstila et al. [1999]).

In the aerosol module, the aerosol density can be either fixed or recomputed at run time according to option `With_fixed_density`. If set to yes the aerosol density will always be equal to the fixed aerosol density mentioned above, if set to no the module will recompute one density for each aerosol bins according to their chemical compositions given by the thermodynamic model.

Two parameterizations are available to compute wet diameters depending on the option `Wet_diameter_estimation`. If set to `Isorropia` the aerosol liquid water content computed by

the thermodynamic model, for instance ISORROPIA, is used. If set to `Gerber`, a simpler but faster method, the Gerber formula, is used.

Note that when `Gerber` option is used, the aerosol density is fixed for all aerosol processes except condensation even if option `With_fixed_density` is set to no. In other words if run time computation of density is chosen, it will only affect condensation. Indeed when using the Gerber formula for fastness purpose there is little interest in recomputing density. Then the fixed density is that specified with `Fixed_aerosol_density` option.

It is possible to use another thermodynamic model instead of ISORROPIA (option `Thermodynamics_module`), but only in a full equilibrium configuration (`Fixed_cutting_diameter` is the maximum diameter, generally 10 µm). The alternative model is EQSAM (Metzger et al. [2002b,a]), version v03d. To obtain the source code of EQSAM, you have to ask Swen Metzger by email (`metzger@mpch-mainz.mpg.de`). EQSAM consists in only one file (`eqsam_v03d.f90`), written in Fortran 90 language. To use EQSAM, you have to put the source code in the directory `include/eqsam/` and compile Polyphemus with `makefile-eqsam.intel` (which only works with the intel compiler).

The *units* of input data (e.g., initial condition, boundary condition etc.) should be given as $\mu g/m^3$ in using `Aerosol_SIREAM_SORGAM` module.

6.3.2 Aerosol_SIREAM_AEC

This aerosol module has been developed on the basis of `Aerosol_SIREAM_SORGAM` module and shares with it most of its characteristics and options. The main difference lies in the replacement of SORGAM organic module by AEC organic module (Pun et al. [2002, 2003]). It also requires a modified version of ISORROPIA, as explained in Section 1.3.6.

Aerosol_SIREAM_AEC We list below the changes between `Aerosol_SIREAM_SORGAM` and `Aerosol_SIREAM_AEC`:

- **Multi-threading**[†1]
  This item about "multi-threading" in the `Aerosol_SIREAM_AEC` module is rather deprecated now. Firstly, because it has also been implemented in `Aerosol_SIREAM_SORGAM`. So, it should not be enlisted as a difference between the two modules. Secondly, as a sophisticated hybrid parallelization is now available for this module, one should not need "old forks" anymore. Nevertheless, it is still available in Polyphemus and should therefore properly documented. The following option enables to use several processors within one machine:

	`[options]`
`Number_of_CPU`	Number of processors to use (see below).

  The whole computation domain will be divided in `Number_of_CPU` sub-domains which will be each solved in parallel by physical processors.

  You can specify any integer value suitable to your machine as `Number_of_CPU`. You can even put a value greater than the number of processors, some of them will just be overloaded.

  You may put a negative integer value to let the module auto-detect the number of processor on your machine. By default then all available processors will be used. Take notice that

[†1] The implementation of this feature slightly differs from what is usually referred as "multi-threading" as we do not use threads but real processes which communicate by shared memory segments.

using this "multithreading" solution in combination with MPI or OpenMP results in an undefined behaviour. When unused, we recommand users to set the value of `Number_of_CPU` to 1.

- **Organic module**
  All physical data needed for organics have been moved from include Fortran files to the `aerosol-species.dat` configuration file.

  The following option lets you the possibility to account for oligomerization (Pun and Seigneur [2007]) of some organic species. Currently only BiA0D (an aldehyde) is concerned.

	`[options]`
`With_oligomerization`	Is oligomerization taken into account?

- **Equilibrium / Dynamic computation**

  This aerosol module can only be run in equilibrium mode, that is to say all bins must be at equilibrium, which means that the `Fixed_cutting_diameter` must always be superior to the upper bound of the aerosol size spectrum.

  Nevertheless one can still choose the dynamic computation but the AEC organic module will be skipped.

- **Missing options**

  Some options of Aerosol_SIREAM_SORGAM have not yet been reported to this module:

  - `Thermodynamics_module`: only `isorropia` is available.
  - `With_cloud_chemistry`: it replaces the previous option. Put yes to take into account cloud chemistry.

The *units* of input data (e.g., initial condition, boundary condition etc.) should be given as $\mu g/m^3$ in using `Aerosol_SIREAM_AEC` module.

6.3.3 Decay

Module `Decay` also supports aerosols. See Section 6.2.4.

Chapter 7

Postprocessing

7.1 Graphical Output

7.1.1 Installation and Python Modules

In Polyphemus, we advocate the use of Matplotlib (with NumPy), Basemap and AtmoPy. Please refer to Sections 1.2 and 1.3 for system-wide installation notes. Below are installation notes for the user.

IPython and Matplotlib

Matplotlib is a Python 2D plotting library which produces high quality figures. On its website http://matplotlib.sourceforge.net/, you may find help for all Matplotlib commands. You may also find many useful examples (Sections "Screenshots" and "Examples (zip)"), a complete tutorial, a useful FAQ, an interesting "cookbook/wiki" and other resources. We highly recommend the use of Matplotlib (actually imported through module `pylab` – see below) together with IPython (enhanced Python shell, http://ipython.scipy.org/) to benefit from a powerful interactive mode.

Once IPython, NumPy and Matplotlib installed, launch IPython with command `ipython`. IPython needs to be launched once to complete its (user) installation. You should get the prompt "`In [1]:`". If you are aware of Python, you can execute Python commands from this prompt.

Now, try to import Matplotlib:

```
In [1]: from pylab import *
```

If no error occurs, your installation is mostly complete. Quit IPython (`ctrl+d` and `RET`). Now edit Matplotlib configuration file. Under Linux or Unix, it is located in `~/.matplotlib/matplotlibrc`. Under Windows, it is in `C:\Documents and Settings\yourname\.matplotlib`. You should find the entries `numerix` and `interactive`. Edit them if necessary (warning: it is case sensitive), so that you have:

```
numerix      : numpy  # numpy, Numeric or numarray
interactive  : True   # see http://matplotlib.sourceforge.net/interactive.html
```

You may change the `backend` depending on what is installed on your computer. Polyphemus development team mostly uses the backend `TkAgg`:

```
backend      : TkAgg
```

But other interactive backends are fine. If you have any question about the backends, consult Matplotlib website.

Now we test the installation. Launch IPython and:

```
In [1]: from pylab import *

In [2]: plot([5, 8])
```

On screen, you should get a new window (a figure) with a line (first diagonal) from $(0, 5)$ to $(1, 8)$. And the prompt should still be available:

```
In [3]:
```

Make another plot:

```
In [3]: plot([6, 7])
```

You should get a new line (in green, probably). The prompt should still be there:

```
In [4]:
```

Basemap

Basemap extends Matplotlib so that one may display fields with a map in the background (World map, Europe map ...).

If your version of Matplotlib is posterior to 0.98, you can test your Basemap installation with:

```
>>> from pylab import *
>>> from mpl_toolkits.basemap import Basemap
>>> m = Basemap(projection = 'cyl')
>>> m.drawcountries()
>>> m.drawcoastlines()
>>> draw()
```

This should show a World map. With a Matplotlib's version prior to 0.98, you should replace the second line with:

```
>>> from matplotlib.toolkits.basemap import Basemap
```

Other examples are available on Matplotlib website.

AtmoPy

Recall that `extract_configuration.cpp` must be compiled (in `atmopy/talos`) – see Section 1.3. Then make sure that Python will be able to find the AtmoPy directory in order to load it. In the distribution, AtmoPy is in directory `Polyphemus/include/atmopy`. When you load AtmoPy under Python (`from atmopy import *` in a program or in IPython), Python searches for the directory `atmopy` in the local directory (that is, `./atmopy`). If `atmopy` is not in the current directory, Python searches for `atmopy` in all paths of `$PYTHONPATH` (under Linux and Unix).

Hence you have two options:

1. copy directory `atmopy` (from `Polyphemus/include/`) into the current directory, or create a symbolic link to it (`ln -s /path/to/Polyphemus/include/atmopy`); or

2. update `$PYTHONPATH` so that it includes the path to `atmopy`, e.g., in Bash: `export PYTHONPATH=$PYTHONPATH:/path/to/Polyphemus/include/atmopy`, which you may put in `~/.bashrc`.

If Matplotlib and Basemap are properly installed, then AtmoPy should work. A test function is provided with AtmoPy to check:

```
>>> from atmopy import *
>>> atmopy_test()
```

If you get Figure 7.1, AtmoPy is properly installed.

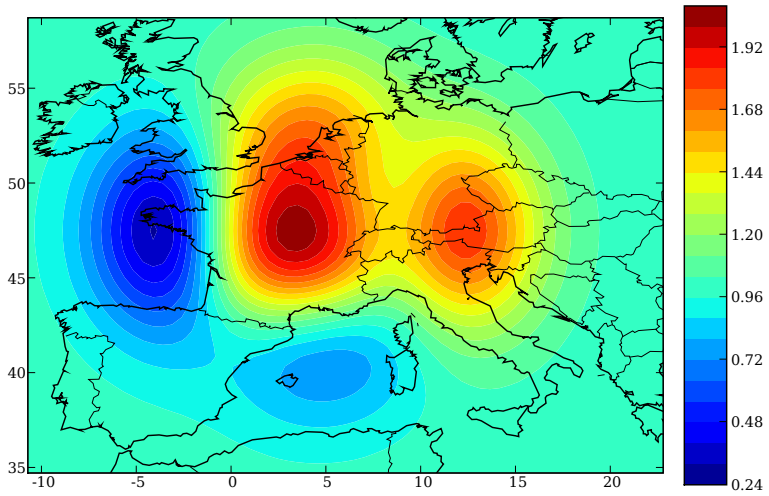


Figure 7.1: Output of function `atmopy_test`. If you get this figure, AtmoPy is properly installed.

7.1.2 A Very Short Introduction to Python and Matplotlib

In this section, several examples are meant to introduce to Python and Matplotlib. The following commands are sometimes given with comments (after character #). You can execute them under IPython if you wish.

Base and Lists

```
>>> x = 5
>>> y = 2 * x + 7
>>> print 2 * y # Under IPython, print is useless; just type 2 * y
34
>>> y = 7; print "y =", y # Combined commands with semicolon
y = 7
>>> a = [-1, 5, 3] # 'a' is a list
>>> print a[0] # 0 is the first index
-1
>>> a = range(10); print a
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> for i in range(3):
...     print "i =", i
i = 0
i = 1
i = 2
```

In Python, blocks are delimited by the *indentation*. For instance (without the prompt >>>):

```
y = 0; x = 0
for i in [2, 5, 15]:    # First loop
    for j in [1, 3, 9]: # Nested loop
        x += i * j      # Inside the inner loop
    y += x              # Outside the inner loop, but still inside the outer loop
print y                 # Outside the loops; the loops are "closed"
```

Arrays (NumPy)

```
>>> from numpy import * # Loads NumPy
>>> a = arange(6, dtype = 'd') # "d" means double (floating point precision)
>>> print a
[ 0.  1.  2.  3.  4.  5.]
>>> a = zeros((2, 3), dtype = 'd') # 2D array
>>> print a.shape
(2, 3)
>>> a[1, 2] = 5.
>>> a[0, :] = 10. # Fills the first line
>>> a[1, 0:2] = -1. # From column 0 to column 1 (2 is excluded)
>>> print a
[[ 10.  10.  10.]
 [ -1.  -1.   5.]]
>>> print a.sum(), a.max(), a.min(), a.mean(), a[:, 0].sum()
33.0 10.0 -1.0 5.5 9.0
>>> print 2. * a[:, 1:] - 2. # Calculation without the first column
[[ 18.  18.]
 [ -4.   8.]]
```

Matplotlib

```
>>> from numpy import *
>>> from pylab import * # Loads Matplotlib
>>> x = arange(10, dtype = 'd')
>>> y = x * x
>>> plot(y)
>>> plot(x, y)
>>> figure() # New figure
>>> plot(x, y, "k-") # "k" for "black", "-" for a solid line
>>> figure(1) # Comes back to the first figure
>>> plot(x, y, "k--") # "--" for a discontinuous line
>>> close() # Closes current figure
```

```
>>> plot(x, y, "k:") # ":" for a dotted line
>>> clf() # Clears the figure
>>> plot(x, y, "k-", label = "Simple") # "label =" is for the legend
>>> plot(x, 2. * y, "k--", label = "Double")
>>> legend()
>>> xlabel("Abscissa")
>>> ylabel("Ordinate")
>>> savefig("plot_example.eps") # Saves the figure in EPS (could be PNG or JPG)
```

7.1.3 Visualization with AtmoPy

AtmoPy provides functions to use Basemap easily and to process data (mainly statistics). It is first used to load binary files (generated in preprocessing, or output of a model or a driver).

Configuration File: `disp.cfg`

In order to load and process data in a binary file, it is convenient to use AtmoPy with a small configuration file, often called `disp.cfg`. This file describes the data to be read:

`[input]` (optional)	
`Nt`	Number of time steps to be read in the binary file. It can be less than the total number of time steps in the file. It cannot be more. If you want to load all available steps, put `0`: `Nt` will be deduced from the file size and other dimensions (`Nx`, `Ny` and `Nz`). If you do so, please check the number of steps that are actually read by AtmoPy; if the number of steps is surprising, check `Nx`, `Ny` and `Nz` in your configuration file.
`x_min`	Abscissa (longitude) of the center of the lower-left cell. It is primarily used to load a background map in figures.
`Delta_x`	Space step along x (longitude). It is primarily used to load a background map in figures.
`Nx`	Number of cells along x.
`y_min`	Ordinate (latitude) of the center of the lower-left cell. It is primarily used to load a background map in figures.
`Delta_y`	Space step along y (latitude). It is primarily used to load a background map in figures.
`Ny`	Number of cells along y.
`Nz`	Number of vertical layers.
`file`	Path to the binary file containing the data.

Here is an example of such a configuration file where the data to be read is in `results/O3.bin` (e.g., ozone at ground level):

```
[input]

Nt = 121
x_min    = -10.0  Delta_x = 0.5  Nx = 67
y_min    = 35     Delta_y = 0.5  Ny = 46
Nz = 1

file: results/O3.bin
```

Note that `general.cfg`, `polair3d.cfg`, ... contain similar entries. You may simply copy and paste these entries. The number of time steps and vertical layers might be different. For instance, `polair3d.cfg` contains the number of model layers, not necessary the number of levels in the target file.

Python Commands: Loading and Processing Data

In IPython, AtmoPy first reads the configuration file (`disp.cfg`):

```
>>> from atmopy import * # Loads AtmoPy
>>> from atmopy.display import * # Loads AtmoPy submodule display
>>> d = getd("disp.cfg") # d is a 4D array
```

You may overwrite the entries in `disp.cfg`:

```
>>> d = getd("disp.cfg", filename = "results/NO.bin") # Loads another file
                                                      # without editing disp.cfg
>>> d = getd("disp.cfg", Nt = 0) # Overwrites Nt
>>> d = getd("disp.cfg", filename = "results/NO.bin", Nt = 0)
>>> d = getd("disp.cfg", filename = "results/NO.bin", Nt = 0, Nz = 2)
```

The array `d` has four dimensions: `Nt`×`Nz`×`Ny`×`Nx`. Hence `d[10, 0, 2, 9]` refers to data at the time step #10 (11^{th} time step since indices start at 0), in the first level, in horizontal cell with indices 2 along y and 9 along x. Another example is `d[15, 0]` which is a 2D array (dimensions: y, x) of data at 16^{th} time step and in the first layer.

A few examples show the way data can be manipulated:

```
>>> d = getd("disp.cfg", filename = "results/O3.bin")
>>> d_ref = getd("disp.cfg", filename = "results/O3-reference.bin")
>>> print d.shape # Same as d_ref.shape: Nt = 48, Nz = 1, Ny = 46, Nx = 67
(48, 1, 46, 67)
>>> print d.mean()
78.5597571528
>>> print abs(d - d_ref).mean()
16.99608674
>>> from numpy import * # Needed for sqrt (see below)
>>> print sqrt(((d - d_ref) * (d - d_ref)).mean()) # Elementwise multiplication
22.6488311576
>>> print (d[10:25] - d_ref[10:25]).min() # Selected time steps
-78.5329427719
>>> print (d[:, 0, 23, 34] - d_ref[:, 0, 23, 34]).max() # Selected cell in the
                                                       # middle of the domain
-1.01527786255
```

Python Commands: Visualization

Using AtmoPy:

```
>>> from atmopy import * # Loads AtmoPy
>>> from atmopy.display import * # Loads AtmoPy submodule display
>>> from pylab import * # Matplotlib is needed for figure() and plot()
>>> d = getd("disp.cfg") # d is a 4D array
```

```
>>> m = getm("disp.cfg") # Loads the background map; an empty figure should pop up
>>> disp(m, d[2, 0]) # Displays data at the third time step and first level
>>> disp(m, d[2, 1]) # Next vertical level
>>> figure() # Another figure
>>> disp(m, d[0, 0], vmin = 0, vmax = 200) # Data range (for the color bar)
>>> dispcf(m, d[10, 0]) # With contours
>>> dispcf(m, d[10, 0], V = 10) # With ten contours
>>> dispcf(m, d[10, 0], V = [0, 50, 100, 150, 200]) # Sets the contours
>>> disp(m, d[10, 0], interpolation = "nearest") # No interpolation
```

Figure 7.1 shows the result of `dispcf` with 25 contours.
Without `disp` or `dispcf` (in case there is no background map, e.g. at small scale):

```
>>> contourf(d[10, 0])
>>> colorbar()
```

Figure 7.2 illustrates `contourf`.

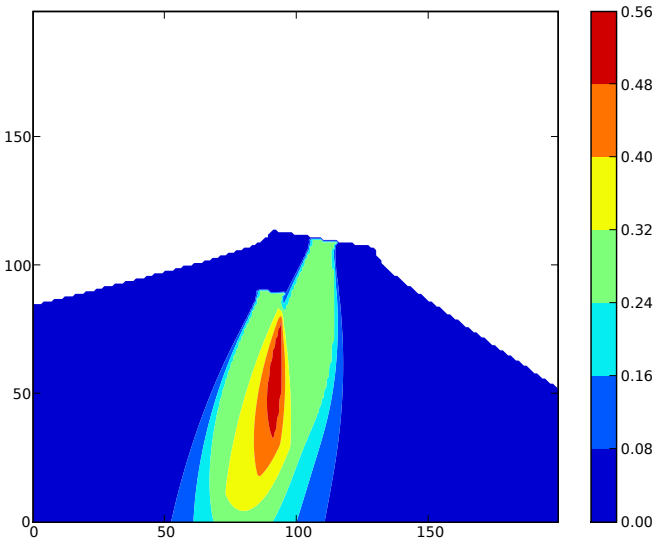


Figure 7.2: Concentration map obtained with the command `contourf`.

In addition, functions `stat.spatial_distribution` and `stat.time_evolution` may be very useful:

```
>>> d = getd("disp.cfg")
>>> print d.shape
(48, 1, 46, 67)
>>> d_max = stat.spatial_distribution(d, "max") # Takes time maxima
>>> print d_max.shape
(1, 46, 67)
```

In every cell, function `stat.spatial_distribution` takes the maximum concentration over the time. If you want to display the time averages:

```
>>> dispcf(m, stat.spatial_distribution(d, "mean")[0])
```

Function `stat.time_evolution` computes the time evolution of a spatial indicator. For instance:

```
>>> d_min = stat.time_evolution(d, "min") # Spatial minimum as function of time
>>> print d_min.shape
(48,)
>>> plot(d_min, label = "Spatial minimum")
>>> plot(stat.time_evolution(d, "max"), label = "Spatial maximum")
```

To Get Further Help

In IPython command line, you can get help this way:

```
>>> help(plot)
>>> help(stat.time_evolution)
```

In addition, all AtmoPy functions are described in a reference documentation (generated with epydoc). See AtmoPy web page: http://cerea.enpc.fr/polyphemus/atmopy.html.

Other online resources:

1. http://diveintopython.org/, learning Python (most useful chapters: 2, 3, 4 and 6);
2. http://docs.python.org/, documentations about Python;
3. http://www.scipy.org/Tentative_NumPy_Tutorial, introduction to NumPy;
4. http://matplotlib.sourceforge.net/, Matplotlib website, see Sections "Tutorial" and "Screenshots";
5. http://www.scipy.org/, SciPy library which includes many scientific modules (linear algebra, optimization, etc.).

7.2 Postprocessing for Gaseous Species

7.2.1 Configuration File

The configuration file `simulation.cfg` is needed for `disp.py` and `evaluation.py`.

Here is a brief explanation of the various options provided in `simulation.cfg` but more details can be found in the file itself for some options.

[input]	
`file`	Binary file with the results to postprocess.
`multiple_file`	Boolean stating whether several files are used.
`t_min`	Initial date of the binary file(s), in format YYYYMMDD or YYYYMMDDHH.
`Delta_t`	Time step in hours.
`Nt`	Number of time step in the binary file(s).
`x_min`	Abscissa of the center of the lower-left cell (longitude in degrees).
`Delta_x`	Step length along x, in degrees (longitude).
`Nx`	Number of cells along x (integer).
`y_min`	Ordinate of the center of the lower-left cell (latitude in degrees).

`Delta_y`	Step length along y, usually in degrees (latitude).
`Ny`	Number of cells along y (integer).
`Nz`	Number of vertical levels (integer).
`station_file`	File describing the stations.
`station_file_type`	Type of station file (Emep, Airbase, BDQA, Pioneer).
`obs_dir`	Directory where observations are stored.
	`[output]`
`station`	Name of the station for which concentrations and observations are displayed.
`t_range`	Dates for which concentrations and observations are displayed.
`concentrations`	What kind of concentrations are displayed: hourly, daily or peak concentrations?
`paired`	Should peak concentrations be paired in time?
`daily_basis`	In case daily concentrations are chosen, are observations provided on a daily basis?
`y_range`	If two numbers are provided, they define the axis range along y (and along x for scatter plots). If only one number is provided, the axis ranges are automatically set.
`scatter`	Is there a scatter plot? See the configuration file for the various options.
`meas_style`	Style for the display of measurements.
`sim_style`	Style for the display of simulated data.
`select_station`	Which stations are involved in statistical measures? Either set to `single` for a single station (defined in `station`), `all` for all stations or a couple *Field-Value* for all stations for which `Field` is equal to `Value`.
`measure`	Statistical measures to be applied to data (see configuration file for all measures available).
`cutoff`	All observations below `cutoff` are discarded.
`ratio`	All stations for which the ratio between the number of available observations and the total number of time steps is below `ratio` are discarded. For instance, if `ratio` is set to `0.3`, stations with over 70% of missing observations are discarded.
`output`	Type of the output (summary, statistics for all stations or results written in a file).
	`[file_list]`
	List of files used if `multiple_file` is set to yes.
	`[legend]`
	List of the legends associated to the files in `[file_list]` (in the same order).

7.2.2 Script `evaluation.py`

Script `evaluation.py` is meant to assess the performances of a chemistry-transport model (CTM). Results of the CTM are compared to measurements at stations and statistics on the differences are computed. The output of the script is presented on screen or can be saved in a

file.

7.2.3 Script `disp.py`

This script written in Python allows to display concentrations and observations at the station `station` with regard to the time. Measurements are displayed with the style defined in `meas_style` and simulation results with `sim_style`.

7.3 Postprocessing for Aerosols

7.3.1 Configuration File

The configuration file `simulation_aerosol.cfg` is the same as `simulation.cfg` (Section 7.2) with aerosol parameters added:

`[input]`	
`Nbins`	Number of size bins.
`computed`	If yes, the bin bounds are computed using a logarithmic law. If no, they are given in a file.
`Dmin, Dmax`	If bin bounds are computed, the minimum and the maximum diameters.
`file_bounds`	If bin bounds are given in a file, the name of the file.
`bin_index_shift`	Number of the first bin (typically 0 or 1).
`primary`	Names of the primary species in the model.
`inorganics`	Names of the inorganic species in the model.
`organics`	Names of the organic species in the model.
`primary_names`	Real names of the primary species (to be displayed).
`inorganics_names`	Real names of the inorganic species (to be displayed).
`output_species`	Aggregated data in output (PM_{10}, $PM_{2.5}$, total mass for each chemical component, total mass and number in each bin).
`with_organics`	If yes, total masses will take into account organic species.
`graph_type`	Graphs that will be displayed when launching `graph_aerosol.py` (chemical composition, mass and number distribution, time series).
`graphs_at_station`	If yes, the graphs will be displayed for the simulation at a given station. If no, graphs will be an average over the domain defined by `i_range` and `j_range`.
`i_range`	First and last indices in x direction for the considered domain.
`j_range`	First and last indices in y direction for the considered domain.
`log_plot`	If yes, the mass and number distributions will be displayed with a log scale for diameters.
`directory_list`	List of directories where outputs are, the aggregated data will be written in a file in the same directory as the output.

The file `simulation_aerosol.cfg` is used by scripts `init_aerosol.py` and `graph_aerosol.py`.

7.3.2 Script `init_aerosol.py`

The outputs of the model for aerosols will be several files: `<species>_<number>.bin` where `<species>` is an aerosol chemical component (in `[aerosol_species]`, see Section 5.5.3) and `<number>` is the index of the size bin. But often, measurements are aggregated data:

- PM_{10} and $PM_{2.5}$ are the mass of aerosol with a diameter smaller than 10 µm and 2.5 µm respectively,
- Total mass of one chemical component.

One can also be interested by the number of particles in each size bin (granulometry), or by the mass distribution along the size bins. This will be done by the script `graph_aerosol.py`, but before you have to launch `init_aerosol.py` by the command:

```
python init_aerosol.py simulation_aerosol.cfg
```

Then you can launch `disp.py` and `evaluation.py` with species such as `PM10`, `PM2.5`, `PNA` (total mass for sodium), etc.

7.3.3 Script `graph_aerosol.py`

You can launch `graph_aerosol.py` by the command:

```
python graph_aerosol.py simulation_aerosol.cfg}
```

Then each desired graphs (specified in `graph_type` section of the configuration file) will be displayed in a different window.

7.4 Computation of Aerosol Optical Parameters

Aerosol optical parameters such as Aerosol Optical Thickness/Depth (AOT/AOD), Single Scattering Albedo (SSA), absorption and extinction coefficients (Babs/Bext) can be computed from the output of a suitable chemistry-transport model. For that purpose, concentrations for each aerosol species in each section and for all the vertical layers should be saved while running polair3d-siream. Details of the equations and the method used are described in Tombette et al. [2008].

First, go to the `optics/` directory. For instance, if you installed Polyphemus in a directory named `Polyphemus/`, you may type:

```
cd Polyphemus/postprocessing/optics/
```

Here are the files `Optics.cxx`, `optics.cpp`, `optics.cfg`, `SConstruct`, `makefile` and the `input/` directory. This code uses data from the OPAC package [Hess et al., 1998] for the indices of species at several wavelengths and tabulations of extinction and absorption efficiencies obtained from the Mie code of W.J. Wiscombe [Wiscombe, 1980].

7.4.1 OPAC Package

The first step is to get the OPAC package. Download the package in the tar file `opac31a.tar.gz` in the `input/` directory from the web address:
`ftp://ftp.lrz-muenchen.de/pub/science/meteorology/aerosol/opac/index.html`.
Untar the file by making:

```
cd input/
tar xzvf opac31a.tar.gz
```

There are two directories resulting from this command: `opac31/` and `optdat/`. Only the files in `optdat/` will be used. As the lines in these files all begin with a #, that Polyphemus does not read, you have to remove all # by making:

```
cd optdat/
../../../../utils/replace_string '#' '' *
```

7.4.2 Tabulation of a Mie Code

The next step is to tabulate the Mie Code. First, you will have to download the routines of the Mie Code. Type:

```
cd ../../../include/
mkdir mie_code_wiscombe
cd mie_code_wiscombe/
```

Then download the files `ErrPack.f`, `MIEV0.f` and `RDI1MACHmodule.F90` from ftp://climate1.gsfc.nasa.gov/wiscombe/Single_Scatt/Homogen_Sphere/Exact_Mie/, and put them in `mie_code_wiscombe/`.

Then go back in the optics postprocessing to compile and execute the tabulation (you may have to change the compiler name in the `makefile` to fit your platform):

```
cd ../../postprocessing/optics/input/Mie_tab/
make
./compute_tab
```

Warning: the tabulation of the efficiency factors will then be done at the wavelength 550 nm, unless you change the value of `lambda0` in `compute_tab.f`. Two files are then created: `Grid_Mie.dat` and `efficiency_factors_tab_550.dat`.

7.4.3 Computation of Optical Parameters

To compute the optical parameters, go to the `optics/` directory and compile:

```
../../utils/scons.py
```

or

```
make
```

The `optics` program generated needs two configuration files: the `general.cfg` from the preprocessing and the `optics.cfg` described hereafter.

	`[paths]`
`Directory_simulation_result`	Path to the results of the polair3d-siream simulation.
`File_temperature`	Path to the temperature field file (general domain).
`File_pressure`	Path to the pressure field file (general domain).
`File_specific_humidity`	Path to the specific humidity field file (general domain).
`Directory_OPAC`	Path to the directory containing the OPAC data (normally `input/optdat/`).
`File_index_water`	Path to the file containing the water refractive indices at several wavelengths (normally `input/index_water_tab.dat`).
`File_species_match`	Path to the file containing the correspondence between the model species and the OPAC species (normally `input/species_opac_match.dat`).

`Directory_efficiency_factor`	Path to the directory containing the efficiency factors file (normally `input/Mie_tab/`).
`Directory_result`	Path to the directory where the computed optical fields will be written.
	`[domain_result]`
`Date`	Date of the beginning of the polair3d-siream simulation.
`t_min`	Starting time (in seconds) since midnight.
`Delta_t`	Simulation time step.
`Nt`	Simulation number of time steps.
`x_min`	Abscissa of the center of the lower-left cell (longitude in degrees).
`Nx`	Number of cells along x (integer).
`y_min`	Ordinate of the center of the lower-left cell (latitude in degrees).
`Delta_y`	Step length along y, usually in degrees (latitude).
`Ny`	Number of cells along y (integer).
`Nz`	Number of vertical levels (integer).
	`[optic]`
`Wavelength`	List of the wavelengths for which the optical parameters are computed (in µm).
`Tabulation_refractive_index_real`	Dimension of the tabulation (efficiency factors) for the real part of the refractive index.
`Tabulation_refractive_index_imaginary`	Dimension of the tabulation (efficiency factors) for the imaginary part of the refractive index.
`Ndiameter`	Dimension of the tabulation (efficiency factors) for the aerosol diameters.
`N_OPAC_wavelength`	Number of wavelengths for which OPAC data are given (do not change).
`N_water_wavelength`	Number of wavelengths for which water refractive indices are given.
	`[aerosol]`
`Nbins`	Number of size bins.
`min_diameter, max_diameter`	The minimum and the maximum diameters.
`aerosol_water_name`	Name of aerosol water content in output.
`Nspecies`	Number of aerosol species in the model (corresponds to the `file_species_match` file).
	`[Options]`
`Dry_diameter_option`	Option to compute the aerosol wet diameter from the dry diameter. Put `1` to use Hänel's relation, `2` for Gerber's formula or `3` to use aerosol water computed in the model.
`Wet_computation_option`	Option to compute the aerosol wet refractive index from the dry refractive index. Put `1` to use Hänel's relation or `2` to use aerosol water computed in the model.

`Well_mixed_computation_option`	Option to compute the index of an internally well-mixed mixture. Put `1` to use the chemical formula or `2` for Lorentz-Lorenz formula.
`Black_carbon_treatment`	Put `2` to consider black carbon as a core or `1` otherwise.

To launch the `optics` program, type:

```
./optics ../../preprocessing/general.cfg optics.cfg 20010101
```

7.5 Ensemble Forecasting

Based on an ensemble of simulations (generated with Polyphemus ensemble capabilities, or generated with other air quality systems), AtmoPy provides methods to produce ensemble forecasts. Assume that, at station s and time round t, the model m predicts $x^s_{m,t}$ and that the corresponding observation is y^s_t. One may try to forecast the time round $T+1$ with a linear combination of the models predictions: $\widehat{y}^s_{T+1} = \sum_{m=1}^{N} v_{m,T+1}\, x^s_{m,T+1}$. The weights $v_{m,T+1}$ of the ensemble forecast $\widehat{y}^s_{T+1}$ are computed with all $x^s_{m,t}$ and y^s_t $(t \leq T)$. The procedure is repeated for all time rounds and is called *sequential aggregation*. For further details, it recommended to read Mallet et al. [2007a] available at `http://www.dma.ens.fr/edition/publis/2007/resu0708.html`. This is done in two steps: (1) the ensemble and the observations are loaded in an instance of `EnsembleData`, and (2) the sequential aggregation is carried out by a derived class of `EnsembleMethod` such as `ExponentiatedGradient` or `RidgeRegression`.

Examples are given in `postprocessing/ensemble/`. You should find the files:

1. `all.cfg`: a configuration file;
2. `example.py`: an example in which the ensemble data is loaded and a few sequential aggregations are performed, including with a meta-learning approach (i.e., the ensemble members, used in the linear combination, already include aggregated predictions);
3. `number_models.py`: tests the performances of an aggregation method against the number of models in the ensemble;
4. `oracle.py`: shows the performances of a posteriori combinations.

7.5.1 Loading Data: Configuration File and `EnsembleData`

`EnsembleData` is a class, defined in AtmoPy, that loads the outputs of an ensemble of models and the corresponding observations. It requires a configuration file like `all.cfg`. Actually `all.cfg` includes more entries than needed by `EnsembleData`: it may also be used with `disp.py` and `evaluation.py` (see Section 7.2). The entries needed by `EnsembleData` are shown below.

`[input]`	
`t_min`	Initial date of the binary files, in format Polyphemus standard format (see Section D.7).
`Delta_t`	Time step in hours.
`Nt`	Number of time step in the binary files.
`x_min`	Abscissa of the center of the lower-left cell (longitude in degrees).
`Delta_x`	Step length along x, in degrees (longitude).
`Nx`	Number of cells along x (integer).

`y_min`	Ordinate of the center of the lower-left cell (latitude in degrees).
`Delta_y`	Step length along y, usually in degrees (latitude).
`Ny`	Number of cells along y (integer).
`Nz`	Number of vertical levels (integer).
`station_file`	File describing the stations.
`station_file_type`	Type of station file (Emep, Airbase, BDQA, Pioneer).
`obs_dir`	Directory where observations are stored.
	`[output]`
`t_range`	Range of dates (standard format, see Section D.7) over which concentrations and observations should be considered.
`concentrations`	What kind of concentrations are considered hourly or peak concentrations?
`paired`	Should peak concentrations be paired in time?
`select_station`	Which stations are involved in statistical measures? Either set to `single` for a single station (defined in `station`), `all` for all stations or a couple *Field-Value* for all stations for which `Field` is equal to `Value`.
`measure`	Statistical measures to be computed (see configuration file for all measures available).
`cutoff`	All observations below `cutoff` are discarded for certain statistical measures.
`ratio`	All stations for which the ratio between the number of available observations and the total number of time steps is below `ratio` are discarded. For instance, if `ratio` is set to `0.3`, stations with over 70% of missing observations are discarded.
	`[file_list]`
	List of binary files that store the ensemble simulations (one file per simulation).

The ensemble may then be loaded with lines like:

```
from atmopy.ensemble import *
ens = EnsembleData("all.cfg", verbose = True)
```

The statistics for all members can be easily accessed:

```
ens.ComputeStatistics()
print ens.stat
```

Even statistics per station, or per time step:

```
ens.ComputeStationStatistics()
print ens.stat_station['rmse'] # or any other measure.
ens.ComputeStepStatistics()
print ens.stat_step['correlation']
```

The main attributes of `ens` are:

1. `Nsim`: number of simulations (i.e., members) in the ensemble;
2. `sim`: list (for simulations) of list (for stations) of arrays (concentrations at a given station);

3. `Nstation`: number of stations;
4. `station`: list of Station instances;
5. `obs`: observations at stations;
6. `date`: dates of observations;
7. `all_dates`: list of all dates in the time period starting with the first observation and ending with the last observation;
8. `stat` (possibly): global statistics;
9. `stat_step` (possibly): statistics per time step.
10. `stat_station` (possibly): statistics per station.

Hence a `EnsembleData` instance gathers all useful information to process an ensemble of simulations and the corresponding observations.

Read AtmoPy reference documentation to get a description of all methods and attributes of a `EnsembleData` instance.

7.5.2 Sequential Aggregation

The sequential aggregation methods, which linearly combine the predictions of an ensemble, are implemented in classes derived from `EnsembleMethod` (also in AtmoPy). The interface of a `EnsembleMethod` (derived) class is similar to that of `EnsembleData`, except that only one member is provided (the linear combination). The attributes are usually:

1. `all_dates`: dates in the covered period;
2. `date`: the list (per station) of dates;
3. `sim`: the ensemble combination;
4. `obs`: corresponding observations;
5. `weight` (if relevant): model weights (indexed by time step if the weights are time-dependent, also indexed by stations if needed);
6. `weight_date` (if relevant): the list (per station) of dates (for weights).
7. `stat` (possibly): global statistics;
8. `stat_step` (possibly): statistics per time step.
9. `stat_station` (possibly): statistics per station.

Applying the sequential aggregation on an instance `ens` of `EnsembleData` is straightforward:

```
em = EnsembleMean(ens) # Trivial combination.
els = ELS(ens) # A posteriori least-squares ensemble.
eg = ExponentiatedGradient(ens) # A learning algorithm.
rg = RidgeRegression(ens) # Another learning algorithm.
rg = RidgeRegression(ens, penalization = 0.001) # With a different parameter.
rg.ComputeStatistics()   # Same as with the
print rg.stat['rmse']    # EnsembleData instances.
```

Consult the AtmoPy reference documentation to get a description of all aggregation methods (from a technical point of view). Read Mallet et al. [2007a], available at `http://www.dma.ens.fr/edition/publis/2007/resu0708.html`, for a complete scientific description. A large number of methods are available; for instance, the list of aggregation methods available in Polyphemus 1.3 is:

1. `BestModel`, `BestModelStep` and `BestModelStepStation`: select the best model according to a given statistics measure, globally, per time step or for each observation (that is, per time step and per station) [a posteriori method];
2. `EnsembleMean`: ensemble mean;
3. `EnsembleMedian`: ensemble median;
4. `ELS`: least-squares ensemble [a posteriori method];
5. `ELSd`: least-squares ensemble per time round [a posteriori method];
6. `ELSdN`: least-squares ensemble per time round, with learning period (also known as "superensemble" [Krishnamurti et al., 2000]);
7. `ExponentiallyWeightedAverage`;
8. `ExponentiatedGradient`, `ExponentiatedGradientWindow` and `ExponentiatedGradientDiscounted`, `ExponentiatedGradientAdaptive`:
9. `Prod`;
10. `GradientDescent`;
11. `RidgeRegression` and `RidgeRegressionDiscounted` and `RidgeRegressionWindow`;
12. `Mixture`;
13. `Polynomial`;
14. `PolynomialGradient`;
15. `FixedShare`;
16. `FixedShareGradient`;
17. `VariableShareGradient`;
18. `OnlineNewtonStep`;
19. `InternalZink`, `InternalPolynomialGradient`, `InternalExponentiatedGradientDiscounted`;
20. `DynamicLinearRegression`.

Also have a look at the example codes `example.py`, `oracle.py` and `number_models.py`.

7.6 Liquid Water Content Diagnosis

The post-processing program `postprocessing/water_plume/water_plume.cpp` uses meteorological data and a concentration field of water (liquid and vapor) and diagnoses the proportion of liquid water. It is launched with two configuration files `water_plume.cfg` and `general.cfg` (which can be merged into a single configuration file) and a date.

7.6.1 Configuration File: `water_plume.cfg`

	`[simulation]`
`Date`	Simulation first day.
`Delta_t`	Simulation time step (in hours).
`PlumeWater`	File containing the simulation results (total water concentration).
`Factor`	Conversion factor to be applied to the water concentration field to convert it into $\mathrm{g\,m^{-3}}$.
	`[meteo]`
`Path`	Path to the meteorological data files.
`Temperature`	Temperature file.
`Pressure`	Pressure file.
`SpecificHumidity`	Specific humidity file.
`LiquidWaterContent`	Liquid water content file.
	`[parameters]`
`source_temperature`	Liquid water potential temperature (in K) at source.
`source_water_content`	Total water content at the source (mass fraction).
	`[output]`
`Option`	Should the liquid water content in the plume only (option `plume`) or in the plume and the ambient air (option `total`) be computed?
`Unit`	Unit of the output. Put `a` for $\mathrm{g\,kg^{-1}}$, and `b` for $\mathrm{g\,m^{-3}}$.
`LiquidWaterContent`	Output file name: file containing the field of liquid water mass fraction.

The water content diagnosis is done at each simulation time step for the whole domain. The domain description is contained in `general.cfg`. Note that you may have to change the number of vertical levels in `general.cfg`, in case not all levels where saved during the simulation.

Appendix A

Polair3D Test-Case

The test case is available on Polyphemus site[†1]. In order to use the test-case, you should download:

- The meteorological data file `MM5-2004-08-09.tar.bz2`. The file is not included in the test-case so that it can be used for various applications and has not to be downloaded each time.
- The archive `TestCase-Polair3D-1.6.tar.bz2`.

Note that you should have Polyphemus installed and working in order to use the test-case.

A.1 Preparing the Test-Case

The first step is to extract the archive `TestCase-1.5-Polair3D.tar.bz2`:

```
tar xjvf TestCase-Polair3D-1.6.tar.bz2
```

The directory `TestCase-Polair3D`, referred to as `TestCase` in what follows, will be created. It is divided in four subdirectories:

- `data`, which contains all precomputed data.
- `raw_data`, which contains all data used for preprocessing. After preprocessing, the results are stored in `data` to be used directly during the simulation.
- `config`, where configuration files are provided.
- `results`, where the results of the simulation are stored.

`MM5-2004-08-09` should be extracted and then placed in `raw_data`.

```
cd TestCase/raw_data/MM5/
wget http://cerea.enpc.fr/polyphemus/test_case/MM5-2004-08-09.tar.bz2
tar xjvf MM5-2004-08-09.tar.bz2
```

Now you have all data necessary to perform preprocessing for the ground and for meteorological data. All other data (emissions, deposition velocities ...) are provided and ready-to-use.

In what follows, ∼`/TestCase` refers to the path to `TestCase-Polair3D` and ∼`/Polyphemus` to the Polyphemus directory path.

[†1] http://cerea.enpc.fr/polyphemus/

A.2 Verifying the General Configuration File

The file `general.cfg` is used by all preprocessing programs and gives a description of the domain and the dates considered. Here is a copy of this file:

```
[general]

Directory_raw_data: raw_data
Directory_computed_fields: data
Directory_ground_data: <Directory_computed_fields>/ground


[domain]

Date: 2004-08-09
Delta_t = 1.0
x_min = -10.0 Delta_x = 0.5 Nx = 65
y_min = 40.5 Delta_y = 0.5 Ny = 33
Nz = 5
Vertical_levels: config/levels.dat
```

Normally this file is written in a way that no modification should be necessary, but you are advised to check it.

Other paths needed for the simulation depend on these ones so modifying them should be sufficient. The domain is defined for a simulation over Europe. Make sure that the date is 2004-08-09 (date for which meteorological raw data is provided).

A.3 Computing Ground Data

Ground data are not necessary to perform the simulation but they are needed to compute the vertical diffusion using Troen and Mahrt parameterization. If you wish to use Louis parameterization, this step is not necessary and you can go to Section A.4.

A.3.1 Land Use Cover

Compile and execute `luc-usgs` (from your `Polyphemus` directory):

```
cd ~Polyphemus/preprocessing/ground/
../../utils/scons.py luc-usgs
cd ~/TestCase/
~/Polyphemus/preprocessing/ground/lus-usgs config/general.cfg config/luc-usgs.cfg
```

The output on screen will be:

```
Reading configuration files... done.
Memory allocation for data fields... done.

Reading LUC data... done.
Building LUC data on output grid... done.

Writing output data... done.
```

A.3.2 Roughness

The preprocessing program `roughness` needs as input data the result of `luc-usgs`.

Compile and execute `roughness`.

```
  cd Polyphemus/preprocessing/ground/
  ../../utils/scons.py roughness
  cd ~/TestCase/
  ~/Polyphemus/preprocessing/ground/roughness config/general.cfg
config/roughness.cfg
```

The output on screen will be:

```
Reading configuration files... done.
Reading roughness data... done.
Writing roughness binary ... done.
```

A.4 Computing Meteorological Data

No modification to configuration file `MM5-meteo.cfg` should be necessary but make sure to use the version of this file included in directory `TestCase` and not in directory `Polyphemus`.

You can open the file and check that `Database_MM5-meteo` is the path to the file `MM5-2004-08-09`.

For details about the other options available in the configuration file, see Section 3.4.5.

Then compile MM5-meteo:

```
  cd ~/Polyphemus/preprocessing/meteo/
  ../../utils/scons.py MM5-meteo
```

and execute it:

```
  cd ~/TestCase/
  ~/Polyphemus/preprocessing/meteo/MM5-meteo config/general.cfg \
config/MM5-meteo.cfg 2004-08-09
```

The output on screen will be:

```
Reading configuration files... done.
Memory allocation for grids... done.
Memory allocation for output data fields... done.
Conversion from sigma levels to heights... done.
Converting from latlon to MM5 indices... done.
Applying transformation to read fields... done.
Computing pressure... done.
Computing surface pressure... done.
Interpolations... done.
Computing Richardson number... done.
Computing attenuation...
  + Computing relative humidity and critical relative humidity... done.
  + Computing cloud profile... done.
  + Computing attenuation... done.
Linear interpolations...
  + Attenuation
```

```
  + SpecificHumidity
  + Liquid Water content
  + CloudHeight
  + SurfaceTemperature
  + SkinTemperature
  + SoilWater
  + SensibleHeat
  + Evaporation
  + SolarRadiation
  + Rain
  + FrictionModule
  + BoundaryHeight
 done.
Computing Kz... done.
Computing PAR... done.
Writing data... done.
```

Note that in that case meteorological data has been generated for 23 hours, but emissions data are only available for this length of time, so it is not necessary to generate more meteorological data.

If you want to compute vertical diffusion using Troen and Mahrt parameterization, compile and execute `Kz_TM`.

```
  cd ~/Polyphemus/preprocessing/meteo/
  ../../utils/scons.py Kz_TM
  cd ~/TestCase/
   ~/Polyphemus/preprocessing/meteo/Kz_TM config/general.cfg \
config/MM5-meteo.cfg 2004-08-09
```

The output on screen will be:

```
Reading configuration files... done.
Memory allocation for data fields... done.

Extracting fields... done.

Computing Kz... done.
Writing output files... done.
```

A.5 Launching the Simulation

A.5.1 Modifying the Configuration File

You should check and modify `polair3d.cfg` if necessary. You have to check the paths (in particular check that the data and saver files are `config/polair3d-data.cfg` and `config/polair3d-saver.cfg`) and to make sure that the date for the simulation is 2004-08-09.

A.5.2 Modifying the Data File

Check `config/polair3d-data.cfg`. If you decided to use Louis parameterization for vertical diffusion, modify the file associated to `VerticalDiffusion` in the section `[meteo]`.

As before, check the paths and dates. In particular, if the dates in any section (except for `[photolysis]`, see below) are not right, you can have an error message.

```
ERROR!
An input/output operation failed in FormatBinary<T>::
Read(ifstream& FileStream, Array<TA, N>& A).
   Unable to read 42900 byte(s). The input stream is empty.
```

Indeed, input data can be computed for several days, so the program will discard the data for the days between `Date_min` in a section of `polair3d-data` and `Date_min` for the simulation. Here, as the data has been computed for one day only, it would be as if the data files were empty, hence this error.

Remark In the case of photolysis, data are provided for a whole year and `Date_min` must be `2004-01-01_12`.

A.5.3 Modifying Saver File

The file `polair3d-saver.cfg` should be ready to use. You can modify the species to save (you are advised against saving concentrations for all species). You can choose to save instantaneous concentrations or concentrations averaged over `Interval_length` by setting `Averaged` to no or yes respectively.

A.5.4 Simulation

Compile the driver.

```
cd  ~/Polyphemus/processing/photochemistry
../../utils/scons.py polair3d
```

Launch the simulation from `TestCase`:

```
cd ~/TestCase/
~/Polyphemus/processing/racm/polair3d config/polair3d.cfg
```

A.5.5 Checking your results

First, check the size of your output files (see 2.7 for details). You can also have a quick look on the values by applying `get_info_float` to for instance `NO.bin`. You should get something like (if you used `Kz_TM` in your simulation chain):

```
Minimum: 3.47425e-09
Maximum: 100.513
Mean: 0.58594
```

A.6 Visualizing Results

A.6.1 Modifying Configuration File

Modify `results/disp.cfg` if necessary (in particular if you have modified `polair3d-saver.cfg`).

```
[input]

# Number of time steps for which concentrations are saved.
Nt = 22
# Domain description for x and y.
x_min    = -10.0     Delta_x = 0.5     Nx = 65
y_min    = 40.5      Delta_y = 0.5     Ny = 33
# Number of levels for which concentration are saved.
Nz = 5

file: O3.bin
```

A.6.2 Using IPython

For details see Section 7.1.3. Remember that the directory `atmopy` should be in your `$PYTHONPATH`. Launch IPython and then type in command line (comments starting with "#"have been added to explain the meaning of each line):

```
cd results/
ipython
  from atmopy.display import *    # Import to the interactive session all
                                  # functions from the module 'display'
                                  # of 'atmopy'
  m = getm('disp.cfg')            # Create the map.
  d = getd('disp.cfg')            # Create a data with the results.
  dispcf(m, d[5,0])               # Display the data for the 8th time step
                                  # and the first vertical level (remember
                                  # that indices start at 0).
```

The image obtained is Figure A.1.

You can create other data if you like to visualize concentrations for other species. In that case, the map has already been created and less information is needed to create the data. In particular it is not necessary to provide a file `disp.cfg`:

```
  d2 = getd(filename = 'NO.bin', Nt = 22, Nz = 5, Ny = 33, Nx = 65)
  disp(m, d2[5,0])
```

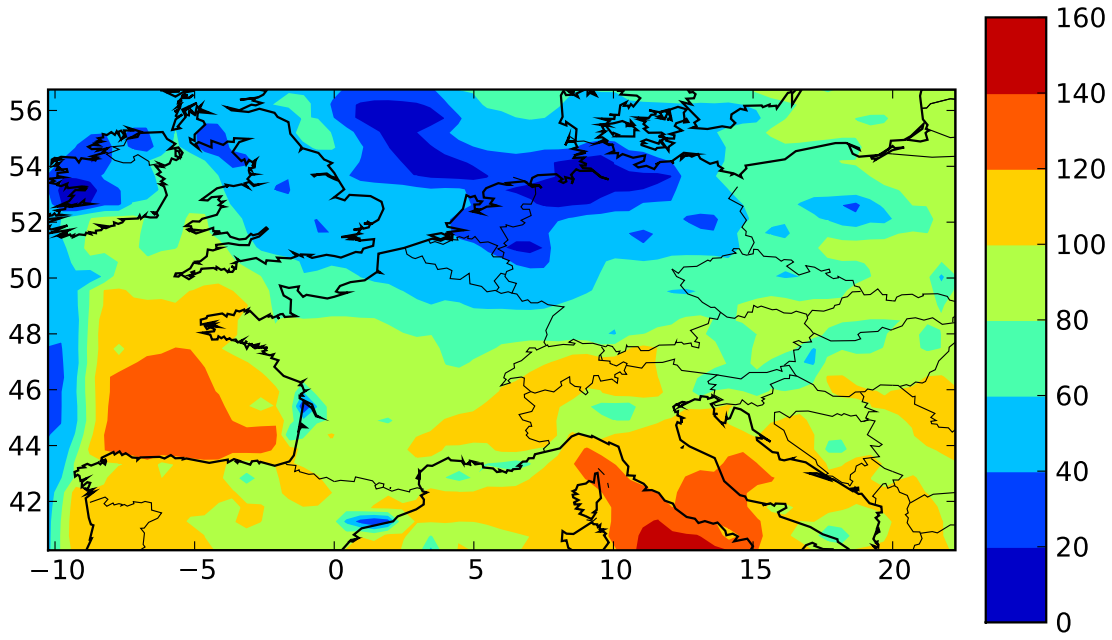


Figure A.1: Figure obtained using IPython and AtmoPy (unit is $\mu g\,m^{-3}$)

Appendix B

Gaussian Test-Case

This document explains how to proceed to perform simulations using the test case for Gaussian models provided with Polyphemus.

When the archive `TestCase-Gaussian-1.8.tar.bz2` is extracted a directory `TestCase-Gaussian/` is created. It is referred to below as `TestCase`.

```
tar xjvf TestCase-Gaussian-1.8.tar.bz2
```

The subdirectory `config/` holds all configuration files necessary and the subdirectory `results/` is meant to store the results of simulations. It is divided in three subdirectories (one for each possible simulation) : `puff_line/` for the Gaussian puff model and a gaseous line source, `puff_aer/` for the puff model with point sources of gaseous and aerosol species, and `plume/` for the Gaussian plume model with gaseous species only.

To launch the test cases, you do not need to modify the configuration files. In the following commands, ∼`/Polyphemus` and ∼`/TestCase` have to be replaced by the paths to the Polyphemus directory and test case directory respectively.

B.1 Preprocessing

Prior to use Gaussian models, you need to compute scavenging coefficients and deposition velocities for the various species. This is achieved by using `gaussian-deposition_aer`.

First compile it :

```
cd ~/Polyphemus/preprocessing/dep/
../../utils/scons.py gaussian-deposition_aer
```

Then run it from the test case directory:

```
cd ~/TestCase/config
~/Polyphemus/preprocessing/dep/gaussian-deposition_aer gaussian-deposition_aer.cfg
```

The output on screen will be :

```
Reading configuration file... done.
Reading meteorological data... done.
Reading species... done.
Reading diameter... done.
```

```
Computation of the scavenging coefficients... done.
Computation of the deposition velocities..done.
Writing data... done.
```

The file `gaussian-meteo_aer.dat` has been created in the directory ∼`/TestCase/config/`. It will be used for puff simulations with aerosol species and with line source.

Note that if your simulation only involves gaseous species, you can use the preprocessing program `gaussian-deposition`. Here we use `gaussian-deposition_aer` because its output can be used for simulations with or without aerosol species.

B.2 Discretization

This step is only necessary for the simulation with a line source. Its aim is to discretize this source into a series of puffs. To do so, compile the preprocessing program `discretization`:

```
cd ~/Polyphemus/preprocessing/emissions/
../../utils/scons.py discretization
```

Then run it from the test case directory:

```
cd ~/TestCase/config
~/Polyphemus/preprocessing/emissions/discretization discretization.cfg
```

The output on screen will be:

```
Reading configuration file... done.
Reading trajectory data...  done.
Length of the trajectory: 48.0278
Number of points on the trajectory: 49
Writing source data... done.
```

The file `puff-discretized.dat` has been created in the directory ∼`/TestCase/config`. It contains a series of puffs representing the discretized line source.

B.3 Simulations

B.3.1 Plume

This simulation uses the program `plume`, which is the program for the Gaussian plume model. It uses the following data:

- Gaseous species : Caesium, Iodine.
- Sources : 2 point sources for Iodine, one point source for Caesium.
- Sources : 2 line sources for Iodine and Biological.
- Meteorological situations : 4 situations, rotating wind with an increasing speed (0.1 $\mathrm{m\,s^{-1}}$, 2 $\mathrm{m\,s^{-1}}$, 5 $\mathrm{m\,s^{-1}}$ and 10 $\mathrm{m\,s^{-1}}$).
- Urban environment.

The simulation uses the following files :

- `plume.cfg` gives the simulation domain, the options and the paths to the other files.
- `gaussian-levels.dat` gives the vertical levels.
- `gaussian-species_aer.dat` gives the species data (species names and radioactive half-lives are used here).
- `meteo.dat` gives all meteorological data. It does not contain scavenging coefficients or deposition velocities since the simulation will not take these processes into account. Therefore, it was not necessary to use the preprocessing program `gaussian-deposition` to create this file.
- `plume-source.dat` contains all the data on stationary sources.
- `plume-saver.cfg` contains the options and paths to save the results.
- `correction-coefficient.dat` contains correction coefficient for the line source Gaussian formula.
- `line-emission.dat` contains the coordinates of the line sources.

Compile the program plume :

```
cd ~/Polyphemus/processing/gaussian
../../utils/scons.py plume
```

Then execute it from ∼`/TestCase/config` :

```
cd ~/TestCase/config
~/Polyphemus/processing/gaussian/plume plume.cfg
```

The output on screen will be :

```
        Temperature     Wind angle      Wind velocity   Stability
Case #0
        10              30              3               A
Case #1
        15              -100            2               B
```

Results are stored in ∼`/TestCase/results/plume/`. You can check the size of the file `Iodine.bin` (see 2.7 for details) and then have a quick look on the values by applying `get_info_float` to `Iodine.bin`. You should get something like:

```
Minimum: 0
Maximum: 3.3259
Mean: 0.0236683
```

B.3.2 Puff with Aerosol Species

The simulation uses `puff_aer`, which is the program for puffs with aerosol species, and the following data:

- Gaseous species : Iodine.
- Aerosol species : aer1, aer2.
- Sources : 1 point source per species.
- Meteorological situation : same 4 situations.
- Rural environment.

The simulation uses the following files:

- `puff_aer.cfg` gives the simulation domain, options and the paths to the other files.
- `diameter.dat` gives the aerosol diameters.
- `gaussian-levels.dat` gives the vertical levels.
- `gaussian-species_aer.dat` gives the species data (only species names are used, since all other data have been used during preprocessing).
- `gaussian-meteo_aer.dat` gives all meteorological data and data on scavenging and deposition. It was created during preprocessing (see Section B.1).
- `puff-source_aer.dat` contains all the data on gaseous and aerosol sources.
- `puff-saver_aer.cfg` contains the options and paths to save the results.

Compile the program `puff_aer`:

```
cd ~/Polyphemus/processing/gaussian
../../utils/scons.py puff_aer
```

Then execute it from `TestCase/config`:

```
cd ~/TestCase/config
~/Polyphemus/processing/gaussian/puff_aer puff_aer.cfg
```

Results are stored in ∼`/TestCase/results/puff_aer/`. You can check the size of your output files (see 2.7 for details) and then have a quick look on the values by applying `get_info_float` to for instance `Iodine.bin`. You should get something like:

```
Minimum: 0
Maximum: 4.83636e+07
Mean: 620.376
```

B.3.3 Puff with Line Source

The simulation uses `puff`, which is the program for puffs with gaseous species only, and the following data:

- Gaseous species : CO2.
- Source : 1 line source.
- Meteorological situations : Same four situations.
- Rural environment.

The simulation uses the following files:

- `puff.cfg` gives the simulation domain, options and the paths to the other files. It also contains the species name.
- `gaussian-levels.dat` gives the vertical levels.
- `meteo.dat` gives all meteorological data. Loss processes are not taken into account so there is no need to have scavenging coefficients or deposition velocities.
- `puff-discretized.dat` gives data on the discretized source. It has been created using program `discretization` (see Section B.2).
- `puff-saver.cfg` gives the options and paths to save the results.

Compile the program `puff`:

```
cd ~/Polyphemus/processing/gaussian
../../utils/scons.py puff
```

Then execute it from `TestCase/config`:

```
cd ~/TestCase/config
~/Polyphemus/processing/gaussian/puff puff.cfg
```

Results are stored in ∼`/TestCase/results/puff_line/`. You can check the size of the file `CO2.bin` (see 2.7 for details) and then have a quick look on the values by applying `get_info_float` to `CO2.bin`. You should get something like:

```
Minimum: 0
Maximum: 6.5739e+07
Mean: 634.202
```

B.4 Result Visualization

To visualize the results of a simulation, use the interactive python interpreter IPython (launched with the command `ipython`). For details see Section 7.1.3.

B.4.1 Gaussian Plume

Launch IPython from the plume results directory:

```
cd ~/TestCase/results/plume
ipython
```

Import the modules that are needed for results visualization with the command:

```
>> import atmopy
>> from atmopy.display import *
```

Then, import the concentration field you want to visualize:

```
>> d = getd(filename = 'Iodine.bin', Nt=4, Nz=2, Ny=200, Nx=200)
```

Nt is normally the number of time steps. Here, as it is a stationary simulation, it should be equal to 1. However, as there are four meteorological situations, we have here $Nt = 4$, as each situation is similar to a time step for an unstationary simulation. It would be the same if it was an unstationary simulation (puff model) with several meteorological situations. If there are 10 time steps, and 4 meteorological situations, you will put $Nt = 40$. The first ten time steps represent the first situation, from 10 to 20 you have the concentration field for the second situation, and so on ...

To visualize the concentration over the domain for the first situation and add a colorbar, use the following commands:

```
>> contourf(d[0,0])
>> colorbar()
```

You should obtain the Figure B.1.

You can visualize the concentration field for the other meteorological situations. You should

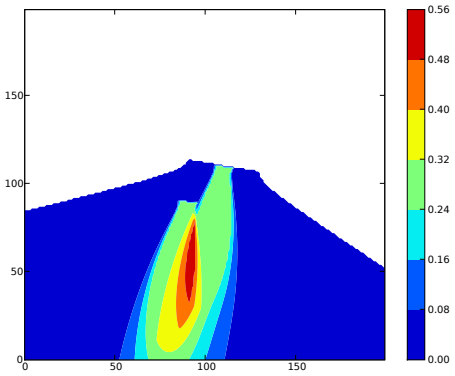


Figure B.1: Plume visualization for the first meteorological situation. Ground concentration in $\mu\,\mathrm{g{\cdot}m^{-3}}$.

see that the wind is turning with increasing speed.

B.4.2 Gaussian Puff with Aerosol Species

You can go into the directory ∼/TestCase/results/puff_aer/ and launch ipython from there, or either change directory from the ipython shell:

```
>> cd ~/TestCase/results/puff_aer/
>>
```

By doing that you ensure that you do not have to import atmopy again. However, when quitting ipython, you will be back in the directory from where it was launched.

To visualize the ground concentration on the domain at time step t for meteorological situation i, you have to visualize the index $(i-1) \times N_t + t$ where N_t is the total number of time steps for one situation (here, $N_t = 80$). For example, the simulation time step 10 for the first situation corresponds to the index 10, for the second situation to the index 90, for the fourth situation to the index 250. To visualize the results at index i,use the command:

```
>> d = getd(filename = 'aer1_0.bin', Nt=320, Nz=2, Ny=30, Nx=55)
>> contourf(d[i, 0])
```

If you want to clear the figure, use the command `clf()`. Figure B.2 gives an example of what you can obtain.

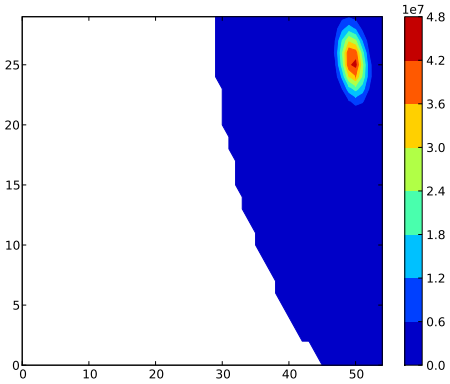
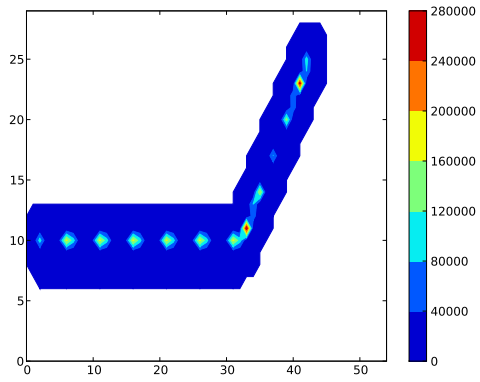


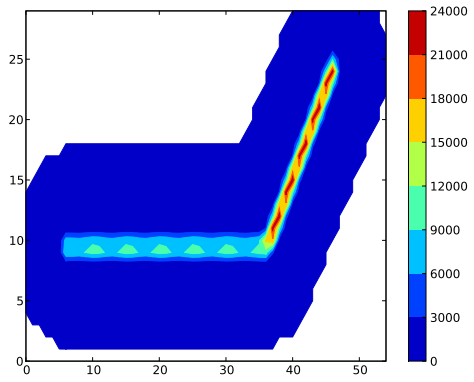
Figure B.2: Puff visualization at time step 30 for species aer1 and first diameter. Third meteorological situation. Ground concentration in $\mu\,\mathrm{g{\cdot}m^{-3}}$.

B.4.3 Gaussian Puff with Line Source

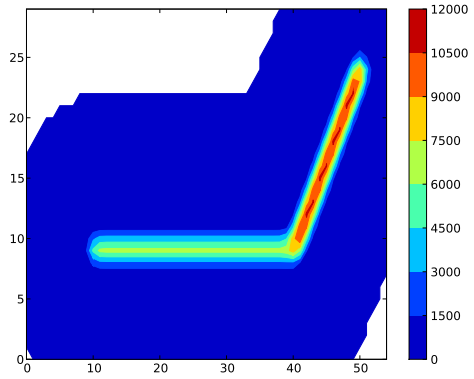
To visualize results, go to the directory ∼/TestCase/results/puff_line/ and use the commands `getd` and `contourf`. Figure B.3 provides examples of what you can obtain.



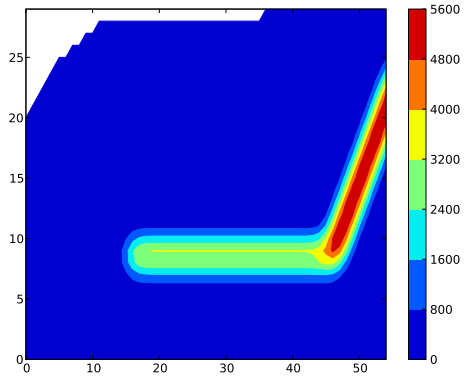
(a) Ground concentration at $t = 1$ s



(b) Ground concentration at $t = 3$ s



(c) Ground concentration at $t = 5$ s



(d) Ground concentration at $t = 8$ s

Figure B.3: Puff line visualization at time steps 10, 30, 50 and 79. Second meteorological situation. Ground concentration in $\mu\,\mathrm{g{\cdot}m^{-3}}$.

Appendix C

Castor Test-Case

The goal of the following test case is to reproduce Chimere test case for the 2003 Heat Wave in Europe.

You must have already downloaded and installed `Polyphemus` to have this test-case working.

First, you need to download `TestCase-1.2-Castor.tar.bz2`, which also works with version 1.8.1 of Polyphemus, from the website. To uncompress this file, execute the following command:

```
$ tar xjvf TestCase-1.2-Castor.tar.bz2
```

This create a directory `TestCase-Castor/` containing:

- a directory `raw_data/`, with data necessary for preprocessing.
- a directory `data/`, for the results of preprocessing.
- a directory `config/`, with all configuration files used.
- a directory `results/`, where the results are stored.
- a program, called `sum-emissions.py`, and its configuration file (`sum-emissions.cfg`), used to sum biogenic and anthropogenic emissions.
- a file version stating for what version of Polyphemus the test-case was made.

The test-case also requires data from Chimere test-case:

- the meteorological file (http://euler.lmd.polytechnique.fr/chimere/downloads/MMOUT_EUR2_20030730_20030803.gz)
- the emission data (http://euler.lmd.polytechnique.fr/chimere/downloads/AemiCONT3-200311.tar.gz)
- the INCA data (http://euler.lmd.polytechnique.fr/chimere/downloads/INCA-200501.tar.gz)

Some command lines have been divided by \ but should be put as one line.

C.1 Modifying the General Configuration File

In what follows, ∼/TestCase refers to the path to TestCase-Castor and ∼/Polyphemus to the path to version 1.5 of Polyphemus.

The file config/general.cfg is used by all preprocessing programs and as such must be the first file you modify when performing preprocessing. Make sure to modify and use the file provided in the directory TestCase/config/.

You should only need to replace the value of <Programs> to have the path to preprocessing in the last version of Polyphemus compatible with the test-case you have. The domain is defined for a simulation over Europe. Make sure that the date is 2003-07-30.

C.2 Computing Input Data

C.2.1 Land Data

A python program is provided among the preprocessing programs to generate land data from Chimere raw data. Files LANDPAR and LANDUSE_CONT3 from Chimere V200606B are necessary to generate land data. They have been included in the archive.

```
$ python ~/Polyphemus/preprocessing/ground/ground-castor.py config/ground-castor.cfg
```

This creates two files (LUC.bin and Roughness.bin) in data/ground/.

C.2.2 Meteorological Data

Download the meteorological file for Chimere test-case and put it in raw_data, then extract it:

```
$ cd raw_data
$ wget http://euler.lmd.polytechnique.fr/chimere/downloads/MMOUT_EUR2_20030730_20030803.gz
$ gunzip MMOUT_EUR2_20030730_20030803.gz
```

After you have done so, execute MM5-meteo-castor to process the MM5 file you have downloaded.

```
$  ~/Polyphemus/preprocessing/meteo/MM5-meteo-castor config/general.cfg \
config/MM5-meteo-castor.cfg 2003-07-30 5d2h
```

The output on screen will be:

```
Reading configuration... done.
Memory allocation for grids... done.
Memory allocation for output data fields... done.
Conversion from sigma levels to altitudes... done.
Converting from latlon to MM5 indices... done.
Computing pressure... done.
Computing surface pressure... done.
Wind rotation... done.
Horizontal interpolations... done.
Vertical diffusion... done.
Computing attenuation... done.
Vertical averages... done.
Writing data... done.
```

This creates 18 binary files in data/meteo/.

C.2.3 Anthropogenic Emissions

We generate anthropogenic emissions using emission data from Chimere test-case. Download the raw data from Chimere website (in the section with old version of the code and old data) and put it in `raw_data`:

```
$ cd raw_data
$ wget http://euler.lmd.polytechnique.fr/chimere/downloads/AemiCONT3-200311.tar.gz
$ tar xzvf AemiCONT3-200311.tar.gz
```

This creates a directory `raw_data/AemiCONT3-200311/`.

Launch the generation of emissions with the following command line:

```
$ ~/Polyphemus/utils/call_dates ~/Polyphemus/preprocessing/emissions/chimere_to_castor \
config/general.cfg config/chimere_to_castor.cfg 20030730 6
```

We use utility program `call_dates` because `chimere_to_castor` can only be launched for one day at a time.

The output for the first day will be:

```
nice time ~/Polyphemus/preprocessing/emissions/chimere_to_castor \
config/general.cfg config/chimere_to_castor.cfg 20030730

Reading configuration... done.
Reading input emissions... done.
Converting to Castor and Polair3D emissions...
   + PPM_big
   + PPM_coa
   + PPM_fin
   + NO
   + NO2
   + HONO
   + CO
   + SO2
   + NH3
   + CH4
   + C2H6
   + NC4H10
   + C2H4
   + C3H6
   + APINEN
   + C5H8
   + OXYL
   + HCHO
   + CH3CHO
   + CH3COE
 done.
```

The program creates 20 binary files in `data/emissions/`.

C.2.4 Biogenic Emissions

Biogenic emissions are generated from meteorological data, using program `bio-castor`. Launch the program with:

```
$ ~/Polyphemus/preprocessing/bio/bio-castor config/general.cfg \
config/bio-castor.cfg 2003-07-30 5d2h
```

The output on screen will be:

```
Reading configuration...
Reading meteorological data... done.
Computing biogenic emissions... done.
Writing output emissions... done.
```

This creates three binary files in `data/bio`: `Isoprene.bin`, `NO.bin` and `Terpenes.bin`.

C.2.5 Summing Emissions

Anthropogenic and biogenic emissions have to be summed. They can be generated for different periods of time, which is why a script has been provided to perform the sum. Launch it with:

```
$ python sum-emissions.py sum-emissions.cfg
```

The output on screen will be:

```
Summing anthropogenic and biogenic emissions.

Anthropogenic species: NO.
Biogenic species: NO.
Computing total emissions for NO.

Anthropogenic species: C5H8.
Biogenic species: Isoprene.
Computing total emissions for C5H8.

Anthropogenic species: APINEN.
Biogenic species: Terpenes.
Computing total emissions for APINEN.
```

This creates three binary files in `data/emissions`: `APINEN-total.bin`, `C5H8-total.bin` and `NO-total.bin`.

C.2.6 Deposition Velocities

Deposition velocities using Emberson parameterization are computed with program `dep-emberson`. Launch it with:

```
$ ~/Polyphemus/preprocessing/dep/dep-emberson config/general.cfg \
config/dep-emberson.cfg 2003-07-30 5d2h
```

The output on screen will be:

```
Reading configuration files... done.
Memory allocation for data fields... done.

Extracting input data... done.
Computing deposition velocities... done.
Writing output data... done.
```

This computes deposition velocities for 23 species.

C.2.7 Boundary Conditions

Download and put in `raw_data/` the INCA files from Chimere test-case.

```
$ cd raw_data
$ wget http://euler.lmd.polytechnique.fr/chimere/downloads/INCA-200501.tar.gz
$ tar xzvf INCA-200501.tar.gz
```

This creates a data directory named `raw_data/INCA`.

First you need to modify the configuration file `config/bc-inca.cfg`. Indeed you have to put the number of time steps for which you want boundary conditions to be generated. As INCA files provide monthly data, you need only to generate the boundary conditions in July for two days, that is to say 48 hours. The configuration file will be:

```
# Configuration file for inca boundary conditions.

[bc_input_domain]

x_min = -180.   Delta_x = 3.75  Nx = 96
y_min = -90.    Delta_y = 2.5   Ny = 73
Nz = 19

# Input species.
Ns = 14
Species: /bc/species_inca.dat


[bc_files]

Nt = 48
Directory_bc: /boundary_conditions/
```

Then launch computation of the boundary conditions with:

```
$ ~/Polyphemus/preprocessing/bc/bc-inca config/general.cfg \
config/bc-inca.cfg raw_data/INCA/INCA.07
```

The output on screen will be:

```
Memory allocation for data fields... done
Reads file... done
Input data processing... done
Species:
```

```
O3 ... done
NO ... done
NO2 ... done
HNO3 ... done
PAN ... done
H2O2 ... done
CO ... done
CH4 ... done
HCHO ... done
C2H6 ... done
NC4H10 ... done
C2H4 ... done
C3H6 ... done
OXYL ... done
```

As the simulation is set in July and August 2003, launch the program again using INCA.08 this time and 78 (hourly) time steps in August.

You will obtain boundary conditions for 14 species.

C.3 Launching the Simulation

C.3.1 Modifying the Configuration Files

You should check and modify `config/castor.cfg` if necessary. You have to check the paths (in particular check that the data and saver file are `config/castor-data.cfg` and `config/castor-saver.cfg`) and to make sure that the date for the simulation is 2003-07-30 (date from which the preprocessing starts).

Then check the paths and dates in `config/data.cfg`. In particular, if the dates in any section are not right, you can have an error message:

```
ERROR!
An input/output operation failed in FormatBinary<T>::
Read(ifstream& FileStream, Array<TA, N>& A).
   Unable to read 42900 byte(s). The input stream is empty.
```

Indeed, input data can be computed for several days, so the program will discard the data for the days between `Date_min` in a section of `polair3d-data` and `Date_min` for the simulation.

Also remember that volume emissions given for NO, APINEN and C5H8 are the sum of anthropogenic and biogenic emissions.

C.3.2 Simulation

Launch the simulation with:

```
$ ~/Polyphemus/processing/castor/castor config/castor.cfg
```

The output on screen will be:

```
Current date: 2003-07-30 00:00
Current date: 2003-07-30 00:10
Current date: 2003-07-30 00:20
Current date: 2003-07-30 00:30
```

```
[...]
Current date: 2003-08-03 23:10
Current date: 2003-08-03 23:20
Current date: 2003-08-03 23:30
Current date: 2003-08-03 23:40
Current date: 2003-08-03 23:50
```

C.3.3 Checking your results

First, check the size of your output files (see 2.7 for details). You can also have a quick look on the values by applying `get_info_float` to, for instance `O3.bin`. You should get something like:

```
Minimum: 0.000103238
Maximum: 134.057
Mean: 52.8842
```

C.4 Visualizing the Results

To visualize the results you have to put the path to `Polyphemus/include` in your `PYTHONPATH`. Then go to the directory `results/` and launch IPython.

```
  $ cd results/
  $ ipython
>>> from atmopy.display import *
>>> m = getm('disp.cfg')
>>> d = getd('disp.cfg')
>>> dispcf(m, d[40, 0])
```

You will obtain the result shown in Figure C.1.

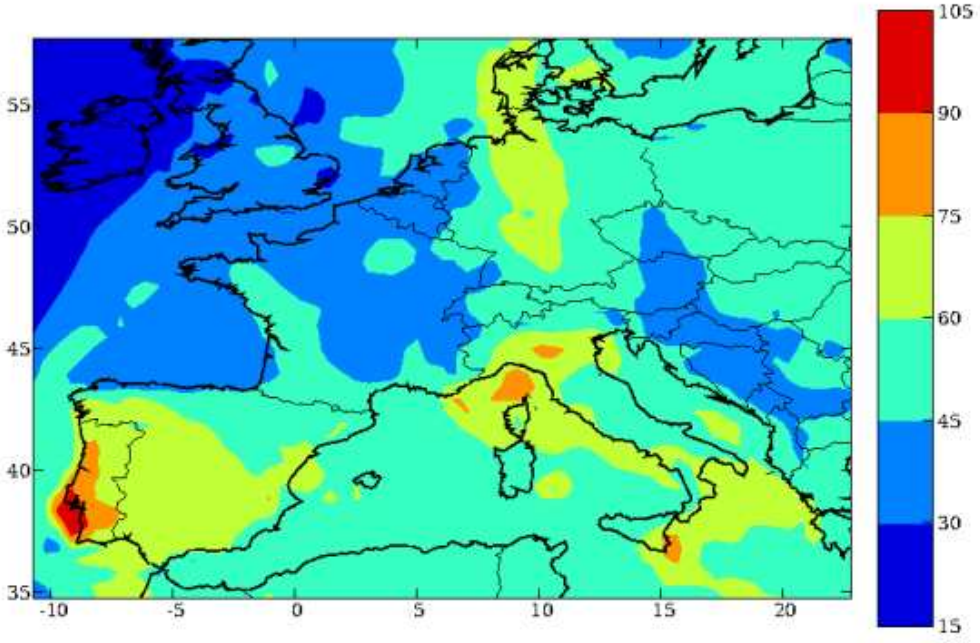


Figure C.1: Figure obtained using IPython and AtmoPy (unit is $\mu g\,m^{-3}$)

Appendix D

Lexical Reference of Polyphemus Configuration Files

D.1 Definitions

All Polyphemus programs rely on flexible configuration files. These configuration files define simulation domains, input and output paths, options, etc.

Configurations files are text files, preferably with extension `.cfg`. They primarily contain *fields*, that is, entries associated with *values* provided by the user. In a configuration file, a line usually reads:

```
field = value
```

A practical example is a discretization definition:

```
x_min = 12.5
Delta_x = 0.5
Nx = 100
```

The *fields* `x_min`, `Delta_x` and `Nx` are associated with proper *values* specified by the user.

The characters put between a field and its value are *delimiters*. In the previous example, the delimiters are blank spaces and equal signs. *Delimiters* are discarded characters. They may be put anywhere in a configuration file but they are always ignored. Their aims are to delimit words (i.e., fields and values) and to make the configuration file clearer.

D.2 Flexibility

The fields and values can be introduced in many ways. First, many delimiters are supported:

- blank space ( ),
- tabulation (	),
- line break,
- equal sign (=),
- colon (:),
- semicolon (;),

- coma (,), and
- vertical bar (|).

For example,

```
x_min = 12.5
Delta_x = 0.5
Nx = 100
```

is equivalent to

```
x_min 12.5
Delta_x == 0.5
Nx: 100
```

Recall that delimiters can only be used to delimit words, and are discarded otherwise. It means that a field or a value cannot contain a delimiter. The fact that the colon is a delimiter may raise a problem under Windows where drives are called `C:`, `D:`, ... In the current version of Polyphemus, full paths (that is, with the drive name) should not be used under Windows. If you need a workaround, please contact the Polyphemus teams at polyphemus-help@lists.gforge.inria.fr.

Fields and values go by pair, but they can be placed anywhere. In particular, several fields may be put on a single line:

```
x_min = 12.5    Delta_x = 0.5    Nx = 100
y_min = -6.2    Delta_y = 1.     Ny = 230
```

The order in which the fields are placed may or may not be important. In most Polyphemus configuration files, the order does not matter. Then

```
x_min = 12.5    Delta_x = 0.5    Nx = 100
y_min = -6.2    Delta_y = 1.     Ny = 230
```

is the same as

```
y_min = -6.2    Delta_y = 1.     Ny = 230
Nx = 100        x_min = 12.5     Delta_x = 0.5
```

Recommandation – Use equal sign '=' between a field and its value if the value is a number and use semi-colon ':' if the value is a string. Example:

```
x_min = 12.5
Output_directory: /home/user/path
```

D.3 Comments

Comment lines may be added. They start with '#' or with '%':

```
# Path where results are written.
Output_directory: /home/user/path
```

They may also be put at the end of a line:

```
Output_directory: /home/user/path  # Path where results are written.
```

Recommandation – Prefer '#' for comments, so as to be consistent with Polyphemus default configuration files.

D.4 Markups

In order to avoid duplications in a configuration file, Polyphemus features a markup management. A markup is denoted with surrounding '<' and '>', e.g. <path>. A markup is automatically replaced with its value whenever it is found. Its value should be provided somewhere in the configuration file with a proper field; for instance, <path> refers to the field `path`. Here is a complete example:

```
Root: /home/user
Input_directory: <Root>/input/
Output_directory: <Root>/output/
```

means:

```
Input_directory: /home/user/input/
Output_directory: /home/user/output/
```

The markup can be used before its value is defined:

```
Input_directory: <Root>/input/
Output_directory: <Root>/output/
Root: /home/user # After calls to <Root>. This is legal.
```

Any field may be used as a markup. The user may define any new markup (that is a new field). Moreover, several markup substitutions can be performed in a single value, and nested markups are properly handled:

```
Home: /home/user
Root: <Home>/Polyphemus/work
Number = 7
Input_directory: <Root>/input-<Number>/
```

is accepted and means:

```
Input_directory: /home/user/Polyphemus/work/input-7/
```

Notice that markups may also replace numbers and may be based on preexisting fields:

```
x_min = 12.5    Delta_x = 0.5    Nx = 100
y_min = <x_min> Delta_y = 1.     Ny = <Nx>
```

D.5 Sections

Fields and values may be protected inside sections. Assume that two domains are defined, one for input data and another for output data. Instead of:

```
x_min_in = 12.5    Delta_x_in = 0.5    Nx_in = 100
x_min_out = 35.8   Delta_x_out = 0.3   Nx_out = 400
```

one may prefer:

```
[input]
x_min = 12.5    Delta_x = 0.5    Nx = 100

[output]
x_min = 35.8    Delta_x = 0.3    Nx = 400
```

Conflicts are avoided and the syntax is clear. This is why most Polyphemus configuration files have sections.

Sections are enclosed by square brackets ('[' and ']').

Markups are not bound to any section.

Do not create a markup with a field which is defined in several section, such as ***x_min*** *in the previous example. Indeed there is no convention on which value of the field to use for markup substitution.*

Recommandation – Put two blank lines before each section and one blank line after:

```
( blank line )
( blank line )
[input]
( blank line )
x_min = 12.5    Delta_x = 0.5    Nx = 100


[output]

x_min = 12.5    Delta_x = 0.5    Nx = 100
```

D.6 Multiple Files

Several Polyphemus programs accept two configuration files as input. Providing two configuration files is then equivalent to providing a single configuration file that would contain all the lines of both files. This is useful to let several programs share a same configuration base. For instance, the simulation domain (whose description is needed by most programs) is usually defined in a configuration file that is provided to every program, in addition to a file dedicated to the specific configuration of the program.

For instance:

```
./emissions general.cfg emissions.cfg 2001-05-06
```

launches the program `emissions` with two configuration files as input: (1) the configuration file `general.cfg` shared with other programs and notably defining the domain description, (2) a specific configuration file, `emissions.cfg`, that includes options for emission generation.

Markups defined in one configuration file can be used in the other file. Note however that each section must be defined in one file only.

D.7 Dates

Date formats are:

```
YYYY                # Year.
YYYY-MM             # With the month.
YYYY-MM-DD          # With the day.
YYYY-MM-DD_HH       # With the hour.
YYYY-MM-DD_HH-II    # With the minute.
YYYY-MM-DD_HH-II-SS # With the second.
```

Months range from `01` to `12`. Days range from `01` to `31`. Hours range from `00` to `23`. Minutes and seconds range from `00` to `59`.

If the month is not specified (format `YYYY`), then the month is set to `01` (January). If the day is not specified (formats `YYYY` and `YYYY-MM`), it is set to `01` (first day of the month). If the hour, the minute or the second is not specified, it is set to zero (`00`).

Hyphens and underscores may be replaced with any character that is neither a delimiter (see Section D.2) nor a cipher. They can also be removed. Examples:

```
19960413
1996-04-13_20h30
1996/04/13@2030
```

Recommandation – Use hyphens around the month and around minutes. Use an underscore between the day and the hour (`YYYY-MM-DD_HH-II-SS`).

D.8 Booleans

Booleans are supported in configuration files and can be specified in any of the following ways:

```
true   t   yes   y
false  f   no    n
```

This is not case-sensitive: e.g., `True` or `NO` are valid.

Bibliography

Arstila, H., Korhonen, P., and Kulmala, M. (1999). Ternary nucleation: kinetics and application to water-ammonia-hydrochloric acid system. *Journal of Aerosol Science*, 30(2):131–138.

Debry, E., Fahey, K., Sartelet, K., Sportisse, B., and Tombette, M. (2007). Technical Note: A new SIze REsolved Aerosol Model (SIREAM). *Atmospheric Chemistry and Physics*, 7:1,537–1,547.

Fahey, K. M. and Pandis, S. N. (2003). Size-resolved aqueous-phase atmospheric chemistry in a three-dimensional chemical transport model. *Journal of Geophysical Research*, 108(D22).

Goliff, W. S. and Stockwell, W. R. (10-12 December 2008). The Regional Atmospheric Chemistry Mechanism, version 2, an update. University of California at Davis. International conference on Atmospheric Chemical Mechanisms.

Hess, M., Koepke, P., and Schult, I. (1998). Optical properties of aerosols and clouds: the software package OPAC. *Bulletin of the American Meteorological Society*, 79(5):831–844.

Horowitz, L. W., Walters, S., Mauzerall, D. L., Emmons, L. K., Rasch, P. J., Granier, C., Tie, X., Lamarque, J.-F., Schultz, M. G., Tyndall, G. S., Orlando, J. J., and Brasseur, G. P. (2003). A global simulation of tropospheric ozone and related tracers: description and evaluation of MOZART, version 2. *Journal of Geophysical Research*, 108(D24).

Krishnamurti, T. N., Kishtawal, C. M., Zhang, Z., T. LaRow, D. B., and Williford, E. (2000). Multimodel ensemble forecasts for weather and seasonal climate. *Journal of Climate*, 13:4,196–4,216.

Louis, J.-F. (1979). A parametric model of vertical eddy fluxes in the atmosphere. *Boundary-Layer Meteorology*, 17:187–202.

Mallet, V., Mauricette, B., and Stoltz, G. (2007a). Description of sequential aggregation methods and their performances for ozone ensemble forecasting. Technical Report DMA-07-08, École normale supérieure de Paris.

Mallet, V., Quélo, D., Sportisse, B., Ahmed de Biasi, M., Debry, É., Korsakissok, I., Wu, L., Roustan, Y., Sartelet, K., Tombette, M., and Foudhil, H. (2007b). Technical Note: The air quality modeling system Polyphemus. *Atmospheric Chemistry and Physics*, 7(20):5,479–5,487.

Metzger, S., Dentener, F., Krol, M., Jeuken, A., and Lelieveld, J. (2002a). Gas/aerosol partitioning: 2. Global modeling results. *Journal of Geophysical Research*, 107(D16).

Metzger, S., Dentener, F., Pandis, S., and Lelieveld, J. (2002b). Gas/aerosol partitioning: 1. A computationally efficient model. *Journal of Geophysical Research*, 107(D16).

Monahan, E. C., Spiel, D. E., and Davidson, K. L. (1986). *Oceanic Whitecaps – and Their Role in Air-Sea Exchange Processes*, chapter A model of marine aerosol generation via whitecaps and wave disruption, pages 167–174. Kluwer Academic.

Nenes, A., Pandis, S. N., and Pilinis, C. (1998). ISORROPIA: A new thermodynamic equilibrium model for multiphase multicomponent inorganic aerosols. *Aquatic Geochemistry*, 4(1):123–152.

Njomgang, H., Mallet, V., and Musson-Genon, L. (2005). AtmoData scientific documentation. Technical Report 2005-10, CEREA.

Pun, B. K., Griffin, R. J., Seigneur, C., and Seinfeld, J. H. (2002). Secondary organic aerosol 2. Thermodynamic model for gas/particle partitioning of molecular constituents. *Journal of Geophysical Research*, 107(D17).

Pun, B. K. and Seigneur, C. (2007). Investigative modeling of new pathways for secondary organic aerosol formation. *Atmospheric Chemistry and Physics*, 7(9):2,199–2,216.

Pun, B. K., Wu, S.-Y., Seigneur, C., Seinfeld, J. H., Griffin, R. J., and Pandis, S. N. (2003). Uncertainties in modeling secondary organic aerosols: Three-dimensional modeling studies in Nashville/Western Tennessee. *Environmental Science & Technology*, 37(16):3,647–3,661.

Rosenbrock, H. H. (1963). Some general implicit processes for the numerical solution of differential equations. *The Computer Journal*, 5:329–330.

Simpson, D., Winiwarter, W., Börjesson, G., Cinderby, S., Ferreiro, A., Guenther, A., Hewitt, C. N., Janson, R., Khalil, M. A. K., Owen, S., Pierce, T. E., Puxbaum, H., Shearer, M., Skiba, U., Steinbrecher, R., Tarrasón, L., and Öquist, M. G. (1999). Inventorying emissions from nature in Europe. *Journal of Geophysical Research*, 104(D7):8,113–8,152.

Smith, M. and Harrison, N. (1998). The sea spray generation function. *Journal of Aerosol Science*, 29:189–190.

Stockwell, W. R., Kirchner, F., Kuhn, M., and Seefeld, S. (1997). A new mechanism for regional atmospheric chemistry modeling. *Journal of Geophysical Research*, 102(D22):25,847–25,879.

Tombette, M., Chazette, P., and Sportisse, B. (2008). Simulation of aerosol optical properties over Europe with a 3-D size-resolved aerosol model: comparisons with AERONET data. *Atmospheric Chemistry and Physics Discussions*, 8:1,321–1,365.

Troen, I. and Mahrt, L. (1986). A simple model of the atmospheric boundary layer; sensitivity to surface evaporation. *Boundary-Layer Meteorology*, 37:129–148.

Vehkamäki, H., Kulmala, M., Napari, I., Lehtinen, K. E. J., Timmreck, C., Noppel, M., and Laaksonen, A. (2002). An improved parameterization for sulfuric acid–water nucleation rates for tropospheric and stratospheric conditions. *Journal of Geophysical Research*, 107(D22).

Wendum, D. (1998). Three long-range transport models compared to the ETEX experiment: a performance study. *Atmospheric Environment*, 32(24):4,297–4,305.

Wesely, M. L. (1989). Parameterization of surface resistances to gaseous dry deposition in regional-scale numerical models. *Atmospheric Environment*, 23:1,293–1,304.

Wiscombe, W. J. (1980). Improved Mie scattering algorithms. *Applied Optics*, 19(9):1,505–1,509.

Yarwood, G., Rao, S., Yocke, M., and Whitten, G. (2005). Updates to the carbon bond chemical mechanism: CB05 final report to the US EPA, RT-0400675. available at: http://www.camx.com/publ/pdfs/CB05_Final_Report_120805.pdf.

Zhang, L., Brook, J. R., and Vet, R. (2003). A revised parameterization for gaseous dry deposition in air-quality models. *Atmospheric Chemistry and Physics*, 3:2,067–2,082.

Zhang, L., Moran, M. D., Makar, P. A., Brook, J. R., and Gong, S. (2002). Modelling gaseous dry deposition in AURAMS: a unified regional air-quality modelling system. *Atmospheric Environment*, 36:537–560.